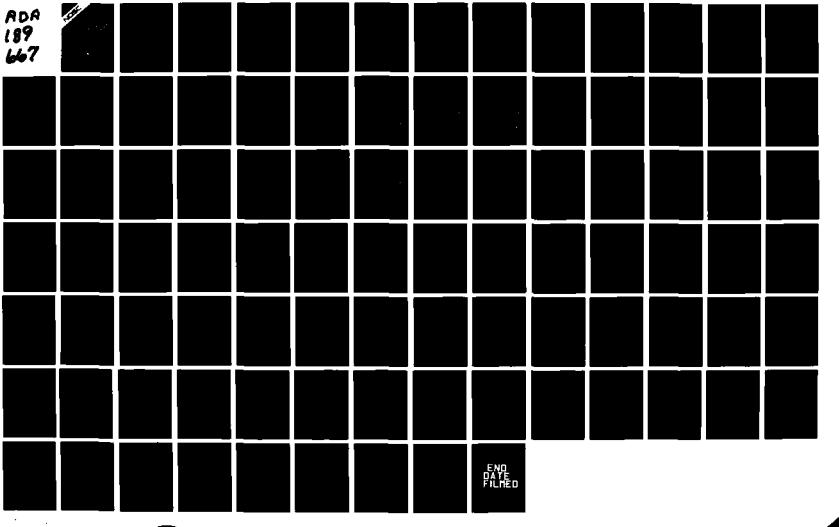
AD A189667

NAVAL OCEAN SYSTEMS CENTER, SAN DIEGO, CA
PERFORMANCE OF DATA COMPRESSION CODES IN CHANNELS WITH ERRORS BY: SARC COMSYSTEMS DIVISION

F/G 17
NOSC TD 1166
UNCLASSIFIED
OCT 1987

ADA
189
667



NAVAL OCEAN SYSTEMS CENTER San Diego, California 92152-5000

NOSC TD 1166

Technical Document 1166
October 1987

Performance of Data Compression Codes in Channels with Errors

SAIC Comsystems Division

NAVAL OCEAN SYSTEMS CENTER

San Diego, California 92152-5000

E. G. SCHWEIZER, CAPT, USN
Commander

R. M. HILLYER
Technical Director

ADMINISTRATIVE INFORMATION

This report was prepared by SAIC Comsystems Division, under contract N66001-85-D-0029, for Code 83 of the Naval Ocean Systems Center.

Released under authority of
W.R. Dishong, Head
Submarine Broadcast Systems Division

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S) NOSC TD 1166
6a NAME OF PERFORMING ORGANIZATION SAIC 6b OFFICE SYMBOL (if applicable)	7a NAME OF MONITORING ORGANIZATION Naval Ocean Systems Center
6c ADDRESS (City, State and ZIP Code) Comsystems Division 2815 Camino Del Rio South San Diego, CA 92108	7b ADDRESS (City, State and ZIP Code) San Diego, CA 92152-5000
8a NAME OF FUNDING SPONSORING ORGANIZATION Space and Naval Warfare Systems Command 8b OFFICE SYMBOL (if applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N66001-85-D-0029
8c ADDRESS (City, State and ZIP Code) Washington, DC 20363-5100	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 33131N; PROJECT NO: CM51; TASK NO: ; AGENCY ACCESSION NO: ICCM 5100

11 TITLE (Include Security Classification)
Performance of Data Compression Codes in Channels with Errors

12 PERSONAL AUTHOR(S)

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Final	FROM Oct 86 TO Jan 87	October 1987	91

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	comma-free codes block codes

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Huffman codes, comma-free codes, and block codes with shift indicators are important candidate message compression codes for improving the efficiency of communication systems. This study was undertaken to determine if these codes could be used to increase the thruput of the fixed very low frequency (FVLF) communication system. This application involves the use of compression codes in a channel with errors.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION	
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ OTIC USERS	UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL W. R. Dishong	22b TELEPHONE (Include Area Code) (619)225-7774	22c OFFICE SYMBOL Code 83

DD FORM 1473, 84 JAN

83 APR EDITION MAY BE USED UNTIL EXHAUSTED
ALL OTHER EDITIONS ARE OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

DD FORM 1473, 84 JAN

TABLE OF CONTENTS

EXECUTIVE SUMMARY iv

Introduction iv
Background iv
Results iv
Findings vi
Conclusions vii
Recommendations vii

INTRODUCTION 1

SCOPE 1

APPROACH 2

DESCRIPTION OF DATA COMPRESSION CODES 3

Generalized Baudot Codes 4
Huffman Codes 5
Comma-free Codes 14

DATA COMPRESSION 21

Introduction 21
Generalized Baudot Codes 26
Huffman Codes 28
Comma-free Codes 32

Surveying Comma-Free Codes 32
Choice of a Comma-free Code for a 58-character Set 38

COMPRESSION CODE PERFORMANCE IN A CHANNEL WITH ERRORS 42

Generalized Baudot Codes 44
Huffman Codes 44

Introduction 44
Description of Simulation Software 44
Simulation Results for a 95-character Set 45
Simulation Results for a 58-character Set 47

Comma-free Codes 63

Comma-free Codes Constructed in Two Steps Using Length One Words 63
Comma-free Codes Constructed Using Other Than Length One Words 75

SUMMARY 77

LIST OF TABLES

E-1. Performance Comparison of Best Data Compression Codes..... v

1. Error Recovery Test using a Five Character Alphabet 12
2. Summary of Narrative Files (IBM PC MASS-11 Files) 23
3. Character Probabilities of Occurrence for Four Narrative Files 24
4. Probabilities of Character Occurrence for any Character in 15 Character Subsets of the Four Narrative Files..... 27
5. Huffman Code Word Lengths for Four Narrative Files 29
6. Average Number of Bits-per-Character for Huffman Codes and Different Training Files 31
7. Partial Survey of the Distributions of Code Word Lengths for Codes Constructed using Prefixes and Suffixes of Lengths 1,1,2 and 3 in the First Three Stages of Construction..... 34
8. Partial Survey of the Distributions of Code Word Lengths for Codes Constructed using Prefixes and Suffixes of Lengths 1,1 and 3 in the First Three Stages of Construction..... 35
9. Partial Survey of the Distributions of Code Word Lengths for Codes Constructed using Prefixes and Suffixes of Lengths 1,1,2 and 4 in the First Three Stages of Construction..... 36
10. Partial Survey of the Distributions of Code Word Lengths for Codes Constructed using Prefixes and Suffixes of Lengths 1,1 and 4 in the First Three Stages of Construction..... 37
11. Huffman and Comma-free Code Word Lengths for Four Narrative Files..... 41
12. Comparison of Bits-per-Character Values of Huffman and Comma-Free Codes..... 43
13. Huffman Code Words for Narrative File IV which Provided the Best Performance in a Channel with Errors 64
14. Probabilities of Erroneous Comma Insertions or Deletions due to Bit Errors for the Suffix-Prefix Comma-Free Code..... 67
15. Probabilities of Erroneous Comma Insertions due to Bit Errors for the Suffix-Suffix Comma-Free Code..... 71

LIST OF FIGURES

1. Two Examples of Huffman Coding..... 6
2. Huffman Codes Include Block Coding for Equal Probability Symbols (8 Symbol Example)..... 8
3. Example Showing Data Compression of a Huffman Code Relative to a Block Code..... 9
4. Examples of Error Propagation for Two Huffman Codes Providing the Same Compression..... 10

LIST OF FIGURES (Cont.)

5. Huffman Code Error Recovery Test using 5-Character Alphabet..... 13
6. An Example of the Construction of a Comma-Free Code..... 15
7. An Example of the Comma-Free Algorithm to Insert "Commas" in an Error-Free Channel..... 16
8. An Example of the Comma-Free Algorithm to Insert "Commas" in a Channel with Errors..... 18
9. Impact of Bit Errors on Comma Placement for Suffix-Prefix Comma-Free Code..... 20
10. Impact of Bit Errors on Comma Placement for Suffix-Suffix Comma-Free Code..... 22
11. Word Lengths for Selected Two- and Three-Step Suffix-Prefix Comma-Free Codes..... 39
12. Word Lengths for Selected Two- and Four-Step Suffix-Prefix Comma-Free Codes..... 40
13. Decoding Error Statistics Resulting from Randomly Induced Bit Errors for a Variety of Huffman Codes..... 46
14. Average Number of Decoded Characters per Bit Error for Four Huffman Codes..... 48
15. Distribution of the Average Ratio of Input Symbol Errors to Bit Errors for a Large Number of Equally Efficient Huffman Codes..... 49
16. Character Decoding Errors for Experiments Using Narrative I as a Training File..... 51
17. Character Decoding Errors for Experiments Using Narrative II as a Training File..... 52
18. Character Decoding Errors for Experiments Using Narrative III as a Training File..... 53
19. Character Decoding Errors for Experiments Using Narrative IV as a Training File..... 54
20. Distribution of Output Errors for Best 58-Character Huffman Code Using Narrative File I as Training File..... 55
21. Distribution of Output Errors for Worst 58-Character Huffman Code Using Narrative File I as Training File..... 56
22. Distribution of Output Errors for Best 58-Character Huffman Code Using Narrative File II as Training File.... 57
23. Distribution of Output Errors for Worst 58-Character Huffman Code Using Narrative File II as Training File.... 58
24. Distribution of Output Errors for Best 58-Character Huffman Code Using Narrative File III as Training File... 59
25. Distribution of Output Errors for Worst 58-Character Huffman Code Using Narrative File III as Training File... 60
26. Distribution of Output Errors for Best 58-Character Huffman Code Using Narrative File IV as Training File.... 61
27. Distribution of Output Errors for Worst 58-Character Huffman Code Using Narrative File IV as Training File.... 62

EXECUTIVE SUMMARY

Introduction

Huffman codes, comma-free codes, and block codes with shift indicators are important candidate message compression codes for improving the efficiency of communication systems. Data compression codes have been used for communications in error-free channels. This study was undertaken to determine if these codes could be utilized to increase the thruput of the fixed very low frequency (FVLF) communication system. This application involves the use of compression codes in a channel with errors.

Background

The investigation of data compression codes was constrained to the investigation of information carrying bits and to compression based on the probabilities of occurrences of characters. The data compression capabilities of the candidate codes were investigated by estimating the average number of bits-per-character for the different codes; the performance of the code in a channel with errors was investigated in terms of the average number of characters decoded in error per bit error and the average number of characters output from the decoder in error per bit error. Generally speaking, as the number of bits-per-character decreases (that is, as data compression increases), the number of characters decoded in error per bit error and the number of characters output from the decoder in error per bit both increase.

Results

The performance of Huffman codes, suffix/prefix comma-free codes, and some variants of Baudot codes were obtained for the encoding of narrative files of an IBM PC for a 58-character set in lieu of processing of Navy messages (which were not available in an IBM PC compatible format). These results should be indicative of the results which could be obtained for the narrative portions of Navy messages using the 58-character (Baudot) set. Huffman code performance results in channels with errors were obtained through simulation on the IBM PC; results for the other codes were obtained analytically.

The number of degrees of freedom in the Huffman code construction process and the complexity of the impacts of bit errors on character synchronization precluded analytical treatment of Huffman code performance in a channel with errors. The severe problems uncovered by the simulation of Huffman codes in these channels led to the consideration of alternative data compression codes less sensitive to bit errors. The error mechanisms for these alternative codes are direct enough to allow analytical treatment.

Table E-1 summarizes the results of this investigation. The comma-free code statistics are for the construction leading

TABLE E-1. PERFORMANCE COMPARISON OF BEST DATA COMPRESSION CODES

COMPRESSION CODE	BITS PER CHARACTER	DECODE CHAR ERR PER BIT ERROR	OUTPUT CHAR ERR PER BIT ERROR
1-shift Baudot	5.1	1.0	1.0
3-shift Baudot	4.5	1.0	1.1
Comma-free	4.0	1.5	1.5
Huffman	3.9	2.2	2.4 *

* A single bit error led to a maximum of 14 output characters for this code.

to a comma-free code most nearly matching the Huffman code in compression. The Huffman code results are for the code constructed with the lowest number of decoded character errors per bit error. The summary results have been rounded to a single significant place to remove the small dependency of the values obtained on the particular narrative used as a basis for estimating character probabilities of occurrence.

Findings

The main findings of this analysis were:

(1) The normal Baudot code uses a single shift key to reduce the number of bits required to transmit information from 6 to about 5.06. Generalizations of this construction can further reduce the average number of bits required to around 4.5 bits-per-character while maintaining a basic block structure.

(2) A suffix/prefix comma-free code can be constructed which provides nearly the same data compression as a Huffman code provided that the probabilities of the occurrence of the different characters decrease in a regular manner. For the character set and probabilities of occurrence of the characters of the set used in the Huffman simulation, the penalty varied from a low of .05 bit per character to a high of .18 bit per character for the four narrative files investigated for using a suffix/prefix comma-free code instead of a Huffman code.

(3) A single bit error can lead to very long sequences of decoding errors when Huffman codes are used. Sequences of output characters in error exceeding 90 characters in length were observed.

(4) It was found that operator interactive processing of the output narrative file could be used to correct about three-quarters of the Huffman decoder errors. Not all the errors could be detected by the operator, given only the output text with errors; some detected errors could not be corrected if multiple bit errors had occurred in the same code word.

(5) In a channel with errors, the performance of the suffix/prefix comma-free codes, which provided the best compression, only depends on whether the code is constructed using a suffix and a prefix or whether it is constructed using two suffixes or two prefixes. The codes in the two categories provide very similar performance in a channel with errors.

(6) A single bit error can lead to at most two character errors for the above prefix/suffix codes. This result follows from the fact that a single bit error can lead to at most one comma being inserted incorrectly by the suffix/prefix comma insertion algorithm for these codes.

(7) A large proportion of the compression gains achievable using Huffman and comma-free codes is provided by the coding of the frequently occurring blank by a short code word.

Conclusions

The following conclusions were drawn as a result of the investigation:

(1) Comma-free codes significantly outperform Huffman codes in an error channel. They provide nearly the same compression and have significantly fewer decoded or output character errors than Huffman codes.

(2) A generalized Baudot code offers modest compression gains (13%) with only one decoded or output character error per bit error.

(3) Comma-free codes probably could be designed with some error correction incorporated into the encoding of end-of-line characters to provide moderate compression gains (around 30%).

(4) More significant compression gains should be achievable by basing either Huffman or comma-free code word assignments to characters on the character and on the one or more characters immediately preceding it in the message.

Recommendations

We make the following recommendations.

(1) The analytical results obtained for the suffix-prefix and suffix-suffix comma-free codes in a channel with errors should be extended to more general comma-free codes.

(2) The most promising codes, i.e., the comma-free codes and the generalized Baudot codes, should be exercised on real Navy messages to verify that the findings reported herein apply to Navy messages.

(3) The most promising codes should be identified to encode characters using their conditional probabilities of occurrence (conditioned on receipt of the one or two previous characters). The techniques developed in this report can be used to identify the best codes for this application.

(4) Error correcting techniques, such as operator interaction or soft-decision logic, should be investigated for use with comma-free codes. Until this has been done, it is difficult to select among the available comma-free codes giving the same compression.

INTRODUCTION

It is desirable to increase the channel capacity of the submarine broadcast system. One technique which has been suggested for doing this is to more efficiently encode the narrative portions of messages through use of data compression codes. A data compression code assigns short binary code words to symbols with a high frequency of occurrence, and long code words to symbols with a low frequency of occurrence. Difficulties arise when data compression codes are used in channels with errors, because one bit error can lead to multiple character errors due to temporary loss of character synchronization.

This report investigates the behavior of Huffman, Comma-free, and generalized Baudot codes for alphabets of 58 characters in channels with errors. Some preliminary results are provided on the feasibility of correcting errors in narrative portions of messages by using narrative context.

SCOPE

Data compression codes have been used in error-free channels, but to our knowledge they have not been used in channels with errors. Consequently, their performance in channels with errors has not been established. This paper represents an initial study of the behavior of data compression codes suitable for encoding the 58-character Baudot code used for Navy messages in a channel with errors.

Results are presented for an alphabet derived from the 95 character set of the IBM PC and for processing narrative files stored on its hard disk. These files were edited to use only capital letters and certain seldom used symbols were deleted, namely " [,], {, }, |", to obtain an alphabet the same size as that required for encoding the Baudot dictionary. The decision to use the reduced IBM PC character set and available document files was made so that results could be obtained without the development of a Navy message data base, which was not available in IBM PC compatible format when this analysis was undertaken.

The analysis is complicated by the fact that the error properties of both Huffman and error-free codes depend on the specific choices of bits and code words, respectively, used to construct the codes for a particular application. This means that codes exist which provide the same data compression gains with differing error properties. The main thrust of this paper is to identify the Huffman codes and the comma-free codes giving the best performance in a channel with errors and compare their performance with that of generalized Baudot codes. This involves characterizing the relationships between character errors and bit errors for the codes.

An additional complication arises in the analysis of comma-free codes: the construction process used does not depend explicitly on the probabilities of occurrence of the characters to be encoded and therefore a given code may not be well matched to the statistics of the character set. An approach is presented which allows the determination of the comma-free code which gives the best data compression, which can be constructed using the procedure developed by R. A. Scholtz. This procedure was used to select the comma-free code which gives the best data compression for the 58-character set used for the Huffman simulations.

APPROACH

Huffman codes are known to provide the best data compression possible for variable length codes. This property is ensured by the code construction process itself which is based directly on the probabilities of occurrences of the characters to be encoded.

The comma-free codes analyzed in this report are known as suffix/prefix codes and are constructed using a sequential procedure found by R. A. Scholtz. This procedure does not utilize probabilities of occurrence to guide the construction process. It was necessary for us to develop an approach to match the word lengths of available prefix/suffix codes to the character probabilities of occurrence to provide comparable data compression to that automatically provided by the Huffman codes.

Even after specifying the distribution of word lengths of Huffman or comma-free codes, there are degrees of freedom in the construction process. It was discovered that the error properties of the codes depended on the choices made in the construction process. This report has been structured to reveal these dependencies and to provide a technique for the selection of the compression codes providing the best performance in a channel with errors.

The insights provided by the investigation of Huffman codes and comma-free codes led to the identification of certain natural extensions of the presently used Baudot codes. A comparison of the performance of these codes with those of Huffman and comma-free codes provides a performance gauge against which the latter codes can be assessed.

The Huffman construction process has a great number of degrees of freedom. The impact of bit errors on character synchronization and character errors is very context-dependent; therefore, an analytical study of the dependency of error statistics on the Huffman construction process could not be performed. A simulation program was written and exercised for many different Huffman codes by altering the specific choices in the Huffman construction process for a fixed character set and fixed probabilities of occurrence for the characters in the character set. The best compression code found in this manner was then further exercised to provide baseline data compression and statistical error properties for Huffman codes.

Unlike the Huffman code, the number of degrees of freedom in the comma-free construction process depends on the number of sequential steps and not on the character set size. The performance of codes constructed in a few steps can be established analytically. Then we found a very surprising thing, the comma-free code which best matches the compression performance of the Huffman codes for the narrative files processed only involved a simple two-step construction. For two-step constructions, the available degrees of freedom for comma-free codes only leads to two code sets with differing error statistics. For these codes the impact of bit errors on both the algorithm which identifies code word (the comma insertion algorithm) and the character decoding process is characterized in terms of the bit within a code word in error.

The remainder of the report is broken into three major sections and a short summary section.

The first major section provides descriptions of the construction processes for generalized Baudot, Huffman, and comma-free codes. Examples of Huffman and comma-free codes are presented to illustrate the dependency of the performance of the codes in a channel with errors on the code construction process. This section provides background and motivation for the remaining sections of the report.

The second major section establishes the data compression which is to be expected for generalized Baudot codes, Huffman codes, and comma-free codes used to encode narrative files based only on estimated probabilities of occurrence of the characters in the narrative files. A symbol set consisting of 58 characters was utilized for this work to best simulate the 58 Baudot character set in use for Navy messages.

The third major section describes the performance of generalized Baudot, Huffman and comma-free codes in a channel with errors.

In the last section, the best codes found are discussed and recommendations submitted.

DESCRIPTION OF DATA COMPRESSION CODES

This section of the report contains three subsections. The first subsection describes a family of compression codes which have a structure very similar to that of the presently used Baudot code. We call these codes generalized Baudot codes. The second and third subsections describe Huffman and comma-free codes, respectively, with emphasis on the description of the construction processes for the codes and their impact on the performance of the codes in channels with errors.

Generalized Baudot Codes

The Navy Baudot code now being used can be viewed as consisting of two kinds of characters: information carrying characters and shift characters. Receipt of a shift character code word changes the decoding of the next code word--the receipt of the shift character by itself does not increase the information passed to the receiver.

The existing Baudot alphabet consists of 57 information characters and one shift character. If a simple block code was used 6 bits would be required. However, if a 5 bit code is used instead, and one of the 32 code words is used as a shift character, 31 information characters can be transmitted using 5 bits and the remaining 26 information characters can be transmitted by using the 5 bit code word reserved for a shift character followed by a 5 bit code word. In effect, the remaining 26 information characters are transmitted using 10 bits.

The shift character can be implemented as either a one-character shift or as a toggle shift. We discuss codes using the shift character as a one-character shift. This is the case amendable to analysis in terms of character probabilities of occurrence.

A simple example suffices to indicate the compression provided by the use of shift characters. Suppose we wanted to encode six characters: a,b,c,d,e, and f, and that the probability of occurrence of a,b, or c was .75 and of the remaining characters .25. If the six characters were encoded with a block code then 3 bits would be required per character. Suppose that "00" was used to transmit "a", "01" for "b", "10" for "c", and "11" a shift character. Then "1100" could be used to transmit "d", "1101" to transmit "e", and "1110" to transmit "f". The average number of bits needed to transmit a character using this code is given by $(.75)(2) + (.25)(4) = 2 + (.25)2 = 2.5$ bits-per-character.

More than one shift character could be used at each stage and more than one shift in succession leading to a whole family of different Baudot-like codes, which we call generalized Baudot codes. For example, suppose again that we are building a code using blocks of two characters. We could reserve two of the 2-bit code words for shift characters. Then we would have two 2-bit code words, and eight 4-bit code words available for encoding information characters. We could use some of the 4-bit code words as shift characters to generate 6-bit code words, and so on.

The data compression provided by any code is determined by the distribution of code word lengths in the code. A generalized Baudot code is specified by its basic code length and the number of characters used as shift characters for each multiple of the

block length. The generalized Baudot codes of most interest for Navy messages use code lengths which are multiples of 3, 4, or 5.

Huffman Codes

Using only the probabilities of a set of characters being transmitted, Huffman provided an organized technique for constructing efficient codes. Huffman codes use the minimum number of bits on the average to transmit characters from the set. The procedure for constructing a Huffman code is illustrated in the following example [reference 1].

Suppose that we wish to code five characters: a, b, c, d, and e with the probabilities 0.125, 0.0625, 0.25, 0.0625, and 0.5, respectively. For this example, which is illustrated in figure 1, the Huffman procedure first involves three regroupings of five characters.

Grouped characters are indicated by (b,d), (a,b,d), and (c,a,b,d) along the top of figure 1. At each stage in this first step, the two characters or group of characters with the lowest probabilities are grouped. A group of characters is assigned the probability obtained by summing the probabilities of the characters in the group.

The Huffman code is constructed based on the characters which have been grouped at each stage by proceeding from right to left. Two of the many possible codes which can be assigned to the original character set are illustrated in figure 1.

We discuss the construction of Code A first. Step 1: assign "0" to the most likely character "e" and "1" to the character set (c,a,b,d). These bits are the first bit in the code words assigned to the characters. The character "e" is distinguished from the characters "c", "a", "b", and "d" by the fact that its code begins with "0" and their codes begin with "1". Step 2: no bit is assigned to "e", and a second bit is assigned to the remaining characters. This bit is chosen to distinguish "c" from "a", "b", and "d"--"0" is shown assigned to "c" and "1" assigned to the other characters. Step 3: no additional bits are assigned to "e" and "c" and additional bits are assigned to distinguish "a" from "b" and "d". Step 4: no bits are assigned to "e", "c", and "a" and bits are assigned to distinguish "b" and "d".

Code B, also shown in figure 1, differs from code A in that at step 1, the character "e" is assigned "1" and the characters "c", "a", "b", and "d" begin with "0". The remaining steps are the same. Note, that "0" and "1" can be assigned in either way at each step, leading to the construction of 16 different codes for the example shown in figure 1.

The example in figure 1 is very regular in that no reordering is necessary during the grouping of characters at the

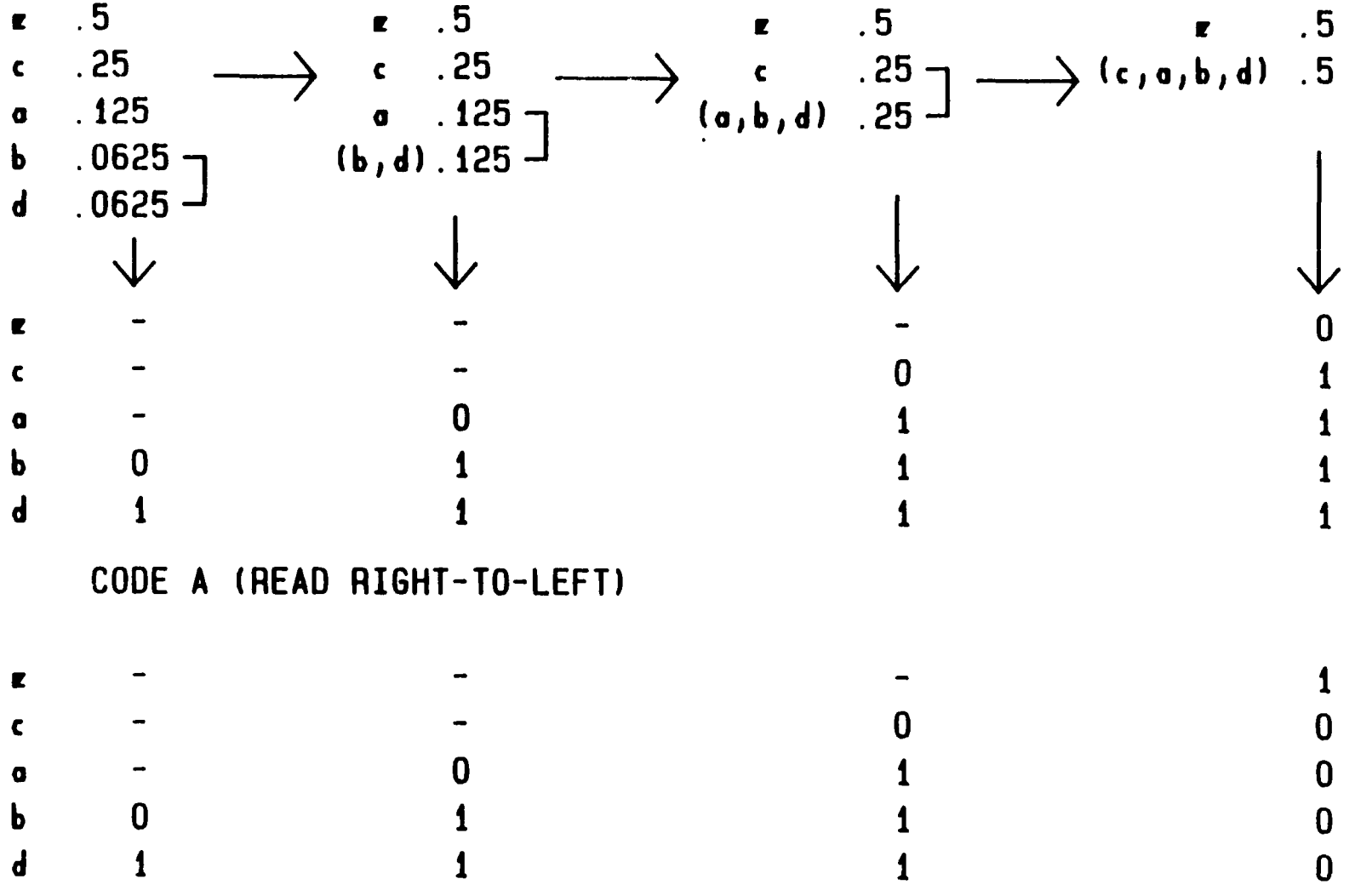


Figure 1. Two Examples of Huffman Coding

different stages of the construction process. It is worthwhile to note that the Huffman coding procedure can lead to block coding when all of the character probabilities are the same.

For example, consider the case of eight characters: a,b,c,d,e,f,g, and h, each having a probability of 0.125. Figure 2 illustrates a Huffman construction process leading to a block code for this case. Note, the characters are listed in the natural alphabet order. The first step leads to grouping g and h, the next step to grouping e and f, the next to grouping c and d, and the fourth step to grouping a and b. Each group is assigned a probability of 0.25. The next two steps leads to grouping e,f,g, and h, and to grouping a,b,c, and d. Each of these groups is assigned a probability of 0.5.

In general, the Huffman code construction process for characters with differing probabilities of occurrence leads to a code with some characters having code words of the same length and other characters having code words of differing lengths.

Figure 3 illustrates the data compression achievable from either code A or Code B (as well as any of the other codes constructable by the Huffman process) described in figure 1. The gain is gauged by comparing the expected average number of bits to transmit a character for the Huffman code with a fixed length code. The average code word length (L) for the example Huffman code is given by:

$$L = 0.125(3) + 0.0625(4) + 0.25(2) + 0.0625(4) + 0.5(1)$$

$$L = 1.875$$

The Huffman code has the smallest average code word length. However, it has variance (V):

$$V = 0.125(3 - 1.875)^2 + 0.0625(4 - 1.875)^2 + 0.25(2 - 1.875)^2 + 0.0625(4 - 1.875)^2 + 0.5(1 - 1.875)^2$$

$$= 1.109375$$

By comparison, Block Coding, which assigns codes of equal length to each symbol, would have produced an average length of 3 with zero variance.

The following examples show how one bit error in Huffman coding can cause errors in more than one character when decoding; extra characters may be introduced or some characters may be dropped. In each case, the first bit of the sequence was changed. The surprising dependency of character errors on the choices made in the Huffman code construction process motivated this study. This phenomenon is illustrated in figure 4.

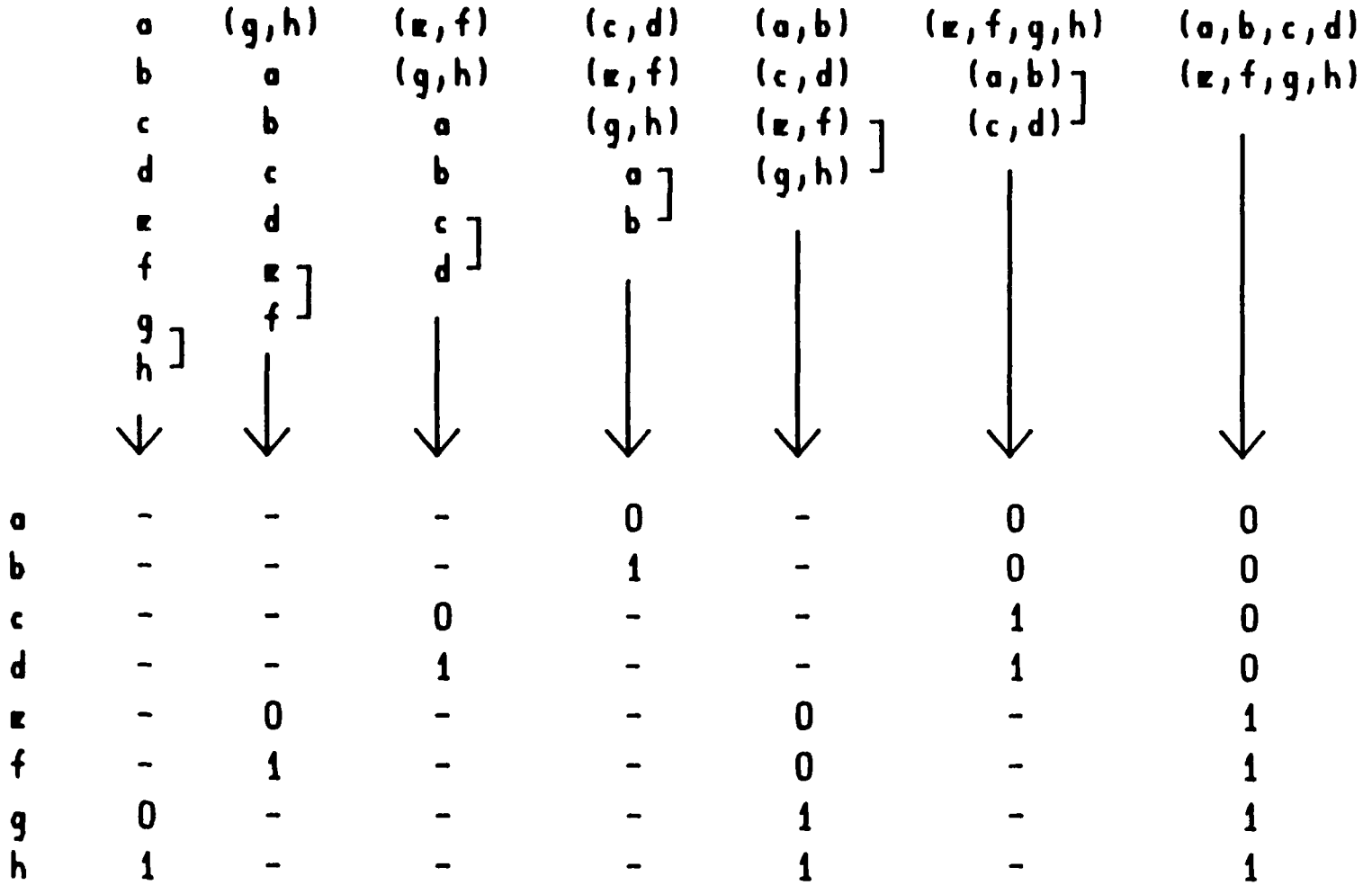


Figure 2. Huffman Codes Include Block Coding for Equal Probability Symbols (8 Symbol Example)

		CODE LENGTH	PROBABILITY		CONTRIBUTION TO BITS/SYMBOL
HUFFMAN CODE	e	1	.5		.5
	c	2	.25		.5
	a	3	.125	→	.375
	b	4	.0625		.25
	d	4	.0625		.25
					1.875
BLOCK CODE	e	3	.5		1.50
	c	3	.25		.75
	a	3	.125	→	.375
	b	3	.0625		.1875
	d	3	.0625		.1875
					3.00

Figure 3. Example Showing Data Compression of a Huffman Code Relative to a Block Code

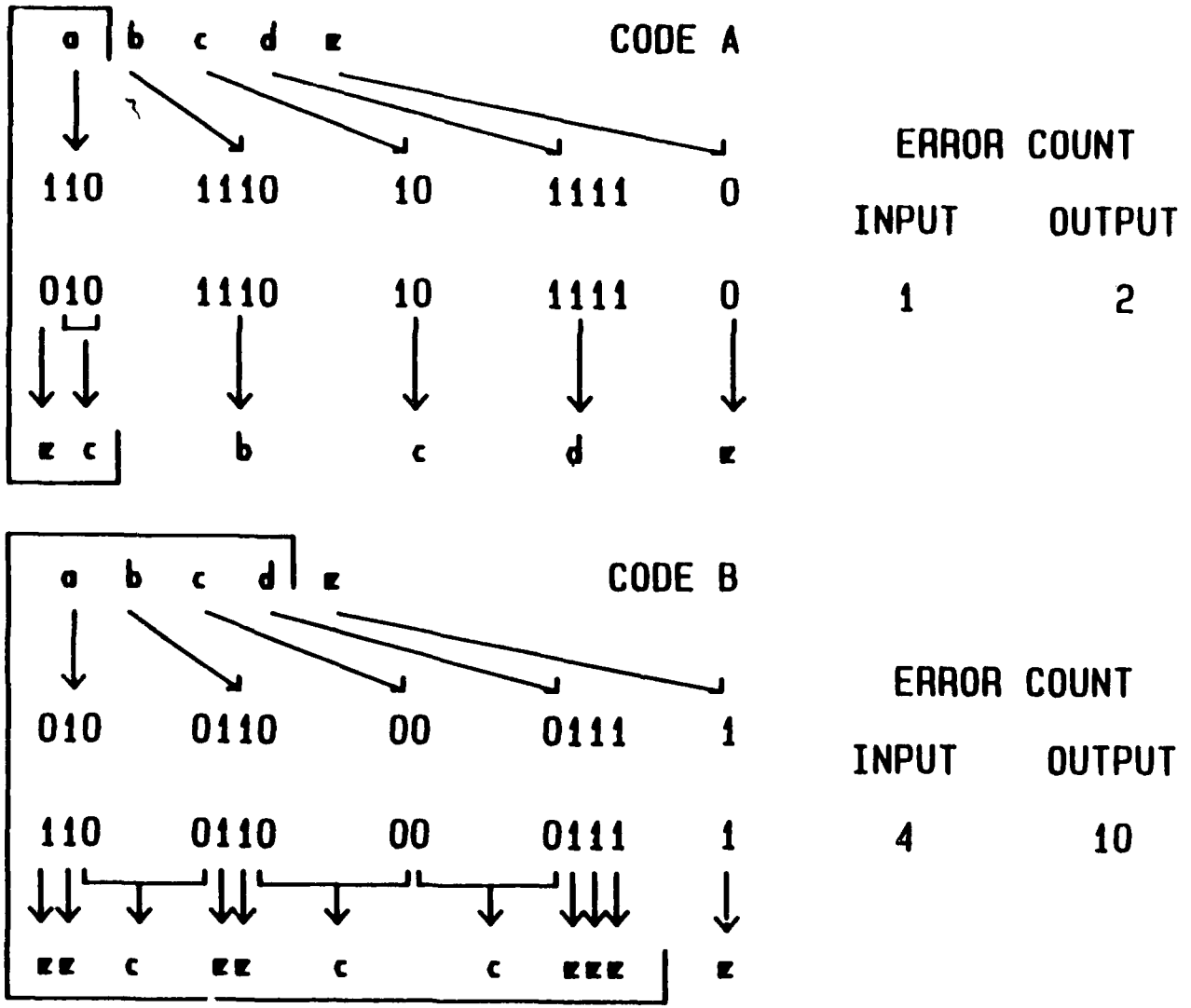


Figure 4. Examples of Error Propagation for Two Huffman Codes Providing the Same Compression

Figure 4 shows the impact of introducing a single bit error into the code word assigned "a" for Code A and Code B. For Huffman codes, and other variable length block codes, the impact of an error depends on the characters following "a". In the example, "abcde" is being transmitted. The impact of the single bit error is enclosed by brackets and an error count shown to the right for each of the two Huffman codes.

For code A, an error in the first bit of the code word for "a" leads to it being incorrectly decoded into the two characters "e" and "c"; i.e., one input character is decoded in error and two erroneous characters are output.

For code B, an error in the first bit of the code word for "a" leads to the next three characters being decoded in error for a total of 10 characters being output erroneously.

For code A, the bit error does not lead to loss of character synchronization; while for code B, it does. In general, bit errors do lead to loss of synchronization for Huffman codes.

Some codes have been discovered which tend to lose character synchronization less often and for shorter periods of time than Huffman codes. These codes utilize an intermediate processing step to define code words (comma insertion) and are called comma-free codes. Comma-free codes are discussed in the next subsection.

The error propagation dependency on Huffman code illustrated by figure 4 was potentially so important that a preliminary simulation was conducted to preclude the possibility that the example was a fluke. The simulations were for the two Huffman codes associated with the example presented in figure 1.

Table 1 summarizes the results of introducing a bit error in the first bit of the first code word. This code word is the encoded first character of the five characters shown in the "input char[acter]" columns of the table; the impact of this bit error on the decoding process is shown by presenting the characters output from the decoder in the "output char[acter]" columns.

Two statistics summarize the experimental results presented in table 1: the number of input symbols decoded in error (2.77 weighted average) and the number of output symbols in error (2.81 weighted average). This second statistic shows on the average how long it takes to regain character synchronization after a bit error is introduced.

The same simulation was run for different Huffman codes obtained by changing the first, second, or third bit of each codeword. The results shown in figure 5 show that there is a very definite dependency of the error properties on the choices made in constructing a Huffman code.

TABLE 1. ERROR RECOVERY TEST USING A FIVE CHARACTER ALPHABET

INPUT CHAR	OUTPUT CHAR	INPUT CHAR	OUTPUT CHAR	INPUT CHAR	OUTPUT CHAR
abcde	eeebcde	bdeac	eebceac	dbcae	eecbcae
abced	eeebced	bdeca	eebceca	dbcea	eecbcea
abdce	eeebdce	beacd	eeaacd	dbeac	eecbeac
abdec	eeebdec	beadc	eeaadc	dbeca	eecbeca
abecd	eeebecd	becad	eeacad	dcabe	eeccabe
abedc	eeebedc	becda	eeacda	dcaeb	eeccaeb
acbde	eeecbde	bedac	eeadac	dcbae	eeccbae
acbed	eeecbed	bedca	eeadca	dcbea	eeccbea
acdbe	eeecdbe	cabde	eceebde	dceab	eecceab
acdeb	eeecdeb	cabed	eceebed	dceba	eecceba
acebd	eeecebd	cadbe	eceedbe	deabc	eeceabc
acedb	eeecedb	cadeb	eceedeb	deacb	eeceacb
adbce	eeedbce	caebd	eceeebd	debac	eecebac
adbec	eeedbec	caedb	eceeedb	debca	eecebca
adcbe	eeedcbe	cbade	ecebede	decab	eececab
adceb	eeedceb	cbaed	ecebeed	decba	eececba
adebc	eeedebc	cbdae	ecebcae	eabcd	ceebcd
adecb	eeedecb	cbdea	ecebcea	eabdc	ceebdc
aebcd	eeeebcd	cbead	eceaad	eacbd	ceecbd
aebdc	eeeebdc	cbeda	eceada	eacdb	ceecdb
aecbd	eeeecbd	cdabe	ececabe	eadbc	ceedbc
aecdb	eeeecdb	cdaeb	ececaeb	eadcb	ceedcb
aedbc	eeeedbc	cdbae	ececbae	ebacd	cebecd
aedcb	eeeedcb	cdbea	ececbea	ebadc	cebedc
bacde	eebecde	cdeab	ececeab	ebcad	cedad
baced	eebeced	cdeba	ececeba	ebcda	cedda
badce	eebedce	ceabd	ebebd	ebdac	cebcac
badec	eebedec	ceadb	ebedb	ebdca	cebcca
baecd	eebeecd	cebad	ebbed	ecabd	cceebd
baedc	eebeedc	cebda	ebbca	ecadb	cceedb
bcade	eedade	cedab	ebcab	ecbad	ccebed
bcaed	eedaed	cedba	ebcba	ecbda	ccebca
bcdae	eeddae	dabce	eecabce	ecdab	ccecab
bcdea	eeddea	dabec	eecabec	ecdba	ccecba
bcead	eedead	dacbe	eecacbe	edabc	cecabc
bceda	eededa	daceb	eecaceb	edacb	cecacb
bdace	eebcace	daebc	eecaebc	edbac	cecbac
bdaec	eebcaec	daecb	eecaecb	edbca	cecbca
bdcae	eebccae	dbace	eecbace	edcab	ceccab
bdcea	eebccea	dbaec	eecbaec	edcba	ceccba

The results presented in table 1 were derived using the following correspondence between characters and code words:

a	<->	110	d	<->	1111
b	<->	1110	e	<->	0
c	<->	10			

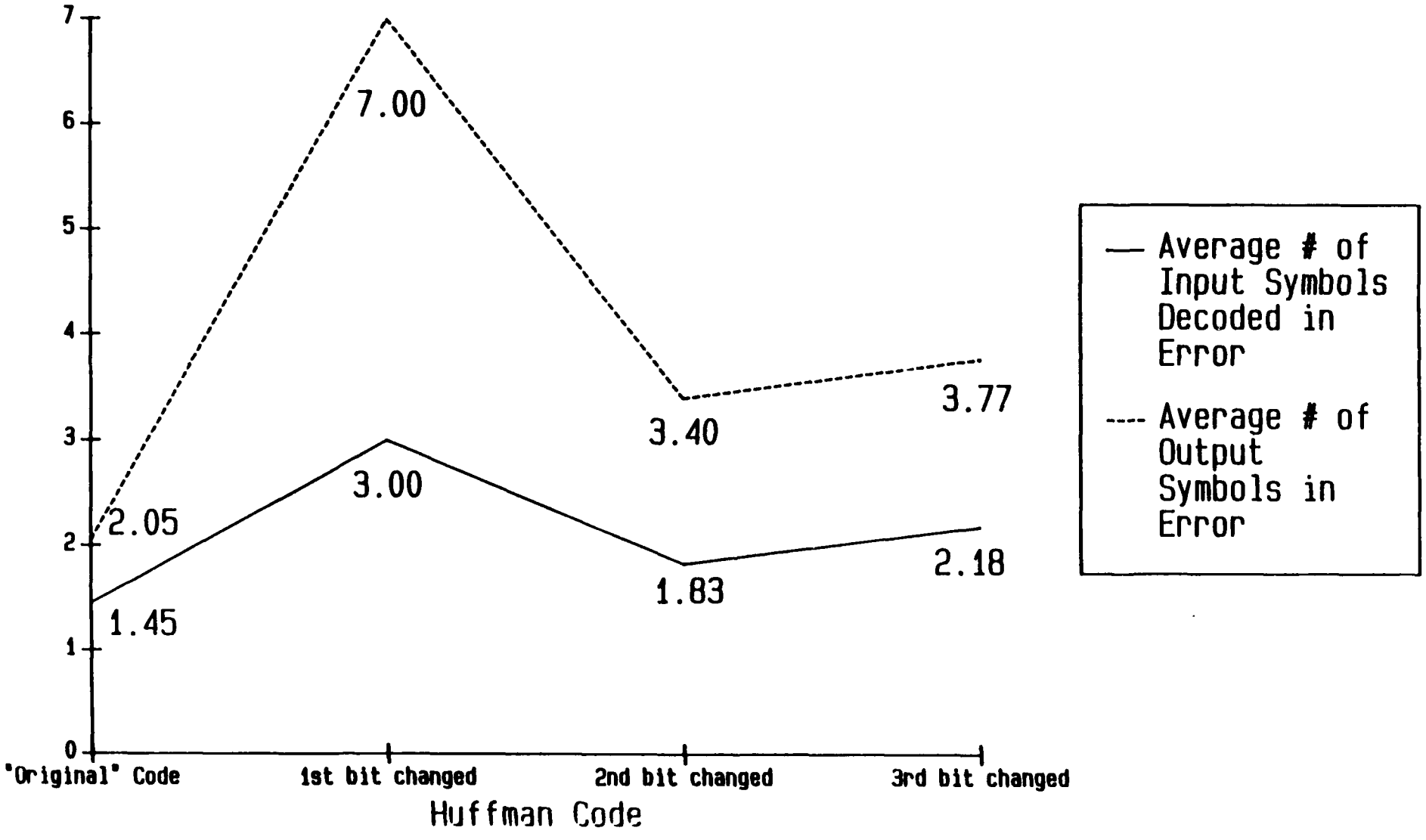


Figure 5. Huffman Code Error Recovery Test Using 5-Character Alphabet

Comma-free Codes

Comma-free codes are binary codes so constructed that it is possible to identify individual code words prior to decoding the received bit stream. In this report, we restrict our attention to a particular family of comma-free codes, known as "suffix/prefix" codes, found by R. A. Scholtz [reference 2].

In order to illustrate the ideas involved in Scholtz's construction process, we choose a particularly simple example derived from a somewhat longer example presented in his paper. Figure 6 illustrates the Scholtz construction process for a code constrained to a maximum code length of five.

Scholtz constructs his code words sequentially. The sets of code words available to be assigned to characters are denoted by "C", "C'", and "C''" in the example. Starting with the set "C", consisting of two code words "0" and "1", the code set "C'" is constructed by taking one of the two original code words and using it as a suffix an arbitrary number of times for the other code word. We chose to use "1" as a suffix and retain "0" as a code word in "C'". Any of the words in "C'" could be used as a suffix to create new code words and thus construct a new set of code words "C''". We choose to use the shortest code word "0" as a suffix to construct "C''". As a result "C''" contains no code words of length one.

We have presented the code words in "C''" in rows according to code word length and by columns beginning with still available code words of "C'". Additional code words could be constructed by choosing, for example, "01" as a suffix, and excluding it as a code word in the new set "C'''" constructed from "C''".

Generally speaking, new code words can be constructed by either using suffixes or prefixes. The process can be carried out any number of times.

Figure 7 illustrates the process used to construct "commas" for the code illustrated in figure 6. Figure 7 shows the comma construction process for an error-free channel and figure 8 illustrates the impact of errors on the construction process.

Suppose that the characters to be transmitted have been assigned the code words shown in brackets in figure 8. The transmitted and received bit stream would consist simply of the bits enclosed in these brackets with no indication of where one code word ended and another began.

Figure 7 shows the three-step process used to insert "commas", i.e., to delineate the code words which were sent. The comma insertion process parallels the code construction process. It proceeds by first inserting commas between all the bits and then successively deleting those according to rules based on the suffix choices.

C: 1, 0

1 AS SUFFIX

C': 0
01
011
0111
01111

0 AS SUFFIX

C": 01

011	010
0111	0100 0110
01111	01000 01100 01110

NOTES:

- THE CONSTRUCTION IS SEQUENTIAL
- A CODE WORD AT ONE STAGE USED AS A SUFFIX CANNOT BE USED AS A CODE WORD FOR THE NEXT STAGE

Figure 6. An Example of the Construction of a Comma-Free Code (Suffix Example Following R. A. Scholtz)

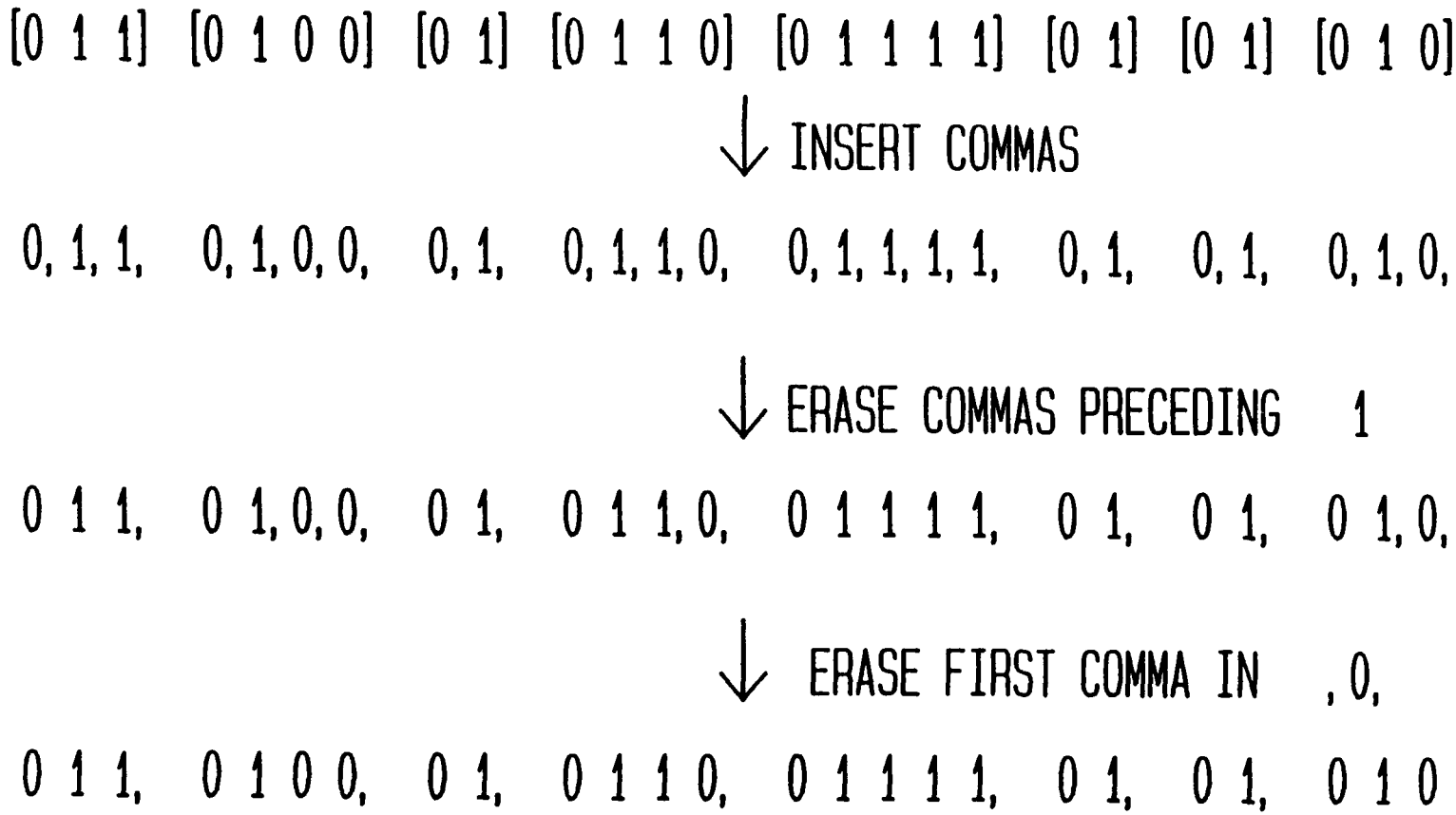


Figure 7. An Example of the Comma-Free Algorithm to Insert "Commas" in an Error-Free Channel (Suffix Example Following R. A. Scholtz)

To aid the reader in following the process, we have maintained the bits in alignment from step-to-step in figure 7--the spaces introduced for this purpose are not interpreted by the decoder. Corresponding to choosing "1" as a suffix, commas are removed preceding "1"s in the second step of the comma insertion process. All the code words in "C'" are now isolated. Next, corresponding to choosing "0" as a suffix to construct "C''", the first comma is removed whenever ", 0," occurs. After the deletion of these commas, all the codes words have been isolated.

Figure 8 traces through the impact of three character errors on the comma insertion process. The transmitted bit stream is the same as that presented in figure 7. The received bit stream shown below the arrow labeled by "errors" has bit errors in the third bit of the first word, the second bit of the fourth word, and the second bit of the next-to-last word. The deletion comma steps leads to the last bit stream. It is easy to see that the leftmost bit error would lead to a character decoding error, but not loss of character synchronization; the second bit error would lead to the previous character and the character with the bit error both being decoded incorrectly, i.e., to lost of character synchronization; the rightmost bit error also leads to the previous character and the character with the bit error being decoded incorrectly and loss of character synchronization.

Even with three bit errors introduced into three of eight characters, three characters were still correctly decoded in the above example. This example shows less impact of errors than previously shown by the Huffman code example.

There are choices in the construction of comma-free codes that would lead to the same distribution of code word lengths, and hence to the same data compression. The behavior of the code in an error channel depends on these choices. This can be illustrated by considering two particularly simple codes that have the same distribution of word lengths. One code is constructed by first using "1" as a suffix and then "0" as a prefix. The second code is constructed by first using "1" as a suffix and then "0" as a suffix, namely the code illustrated for up to length five code words in figure 6. The first code will be referred to as the suffix-prefix code and the second code as the suffix-suffix code. (These codes turn out to be very important for practical applications; this will be discussed in a later section.)

The code words of either of the two codes have lengths 2 to $m > 1$. The code words are easy to describe mathematically:

(a) suffix-prefix code words are of the form

k "0"s followed by h "1"s with $k > 0$, $h > 0$, $k + h <= m$

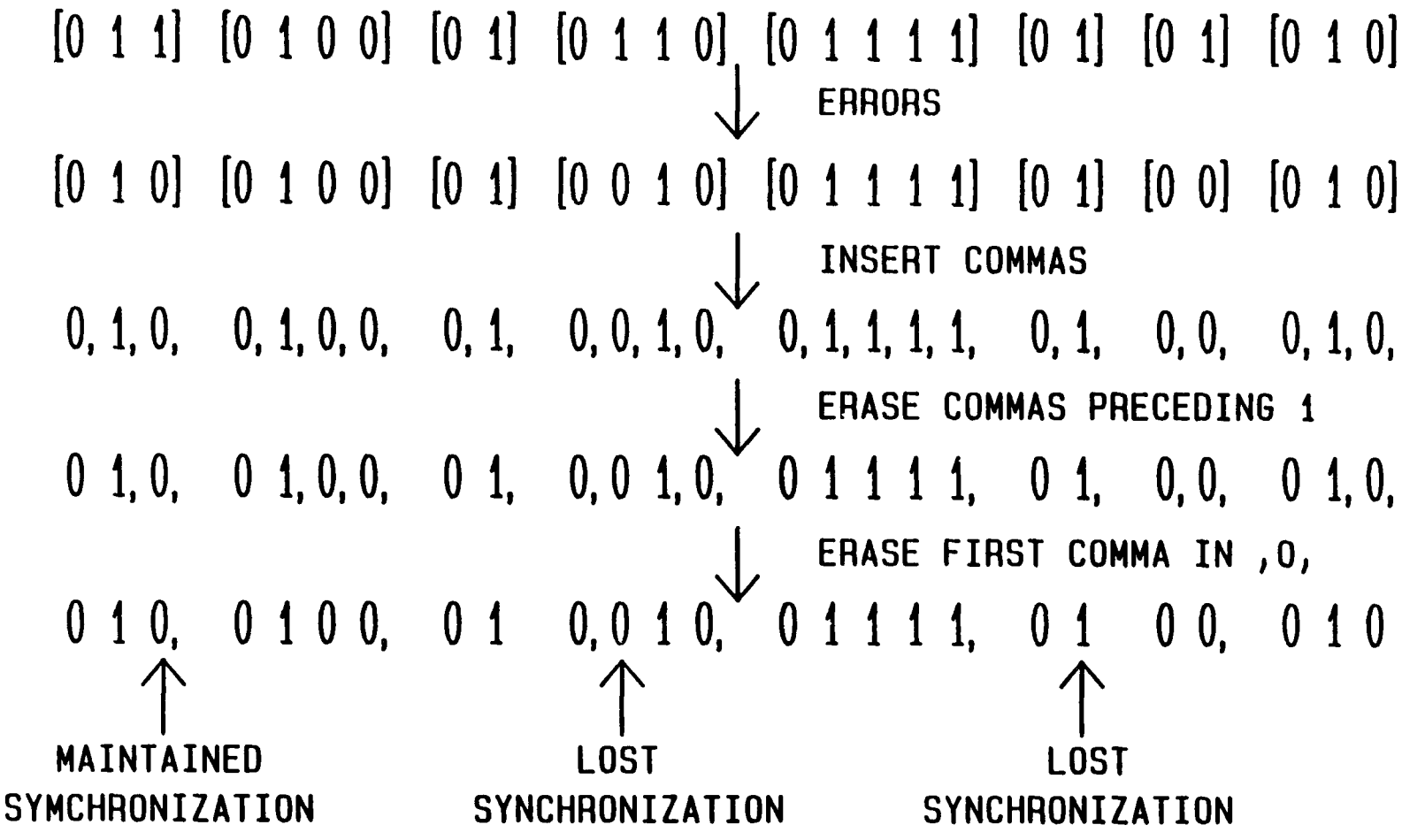


Figure 8. An Example of the Comma-Free Algorithm to Insert "Commas" in a Channel with Errors

(b) suffix-suffix code words are of the form

1·"0" followed by k "1"s followed by h "0"s with
k > 0, h >= 0, 1 + k + h <= m

It is particularly easy to describe the impact of bit errors on the comma insertion process of the suffix-prefix code. Figure 9 summarizes the impact of bit errors on the process as a function of where the bit error occurs in a code word bracketed by two other code words. Four cases are distinguished: first bit in error, either of the two transition bits from "0" to "1" in error, last bit in error, and a central "0" or "1" bit in error.

The comma insertion process can lead to code words which are longer than m bits and therefore not decodable. If the first bit is in error, and k2 > 1, then this occurs only if the first word has maximal length m; if the last bit is in error, and h2 > 1, then this occurs only if the last word has maximal length m. In general, if the first bit is in error and k2 > 1, a comma is inserted (incorrectly) one position to the left; if the last bit is in error and h2 > 1, a comma is inserted (incorrectly) one position to the right. If the first bit is in error and k2 = 1, then the comma between the first and second word is deleted leading to a code word of the length k1 + h1 + 1 + h2. The new word being non-code word if this expression exceeds m. Likewise, if the last bit is in error and h2 = 1, the comma between the second and third word is deleted leading to a code word of length k2 + 1 + k3 + h3, which can be non-code word if this expression exceeds m.

Errors in the transition bit, namely the k2-th bit or the (k2+1)-th bit, with k2 > 1 and h2 > 1, does not impact the positions at which commas are inserted. The resulting erroneous word is always decoded as a single character.

Errors in the middle of a string of "0"s or in the middle of a string of "1"s lead to the insertion of a spare comma within the middle word. The middle word is always decoded as two characters for these cases.

In summary, this survey of the impact of bit errors on the suffix-prefix comma insertion and decoding process has shown that a single bit error can lead to at most one comma being inserted incorrectly. All the following possibilities occur: the comma between the first two words can be deleted or moved to the right, the comma between the second and third words can be deleted or moved to the left, or a new comma can be inserted splitting the middle word. However, it can happen the commas are all inserted correctly and only a decoding error occurs. A single bit error leads to either one or two character errors so that, unlike Huffman codes, the impact of a single bit error on character decoding is strictly limited.

A discussion of the impact of bit errors on the suffix-suffix code is somewhat more complicated than for the suffix-

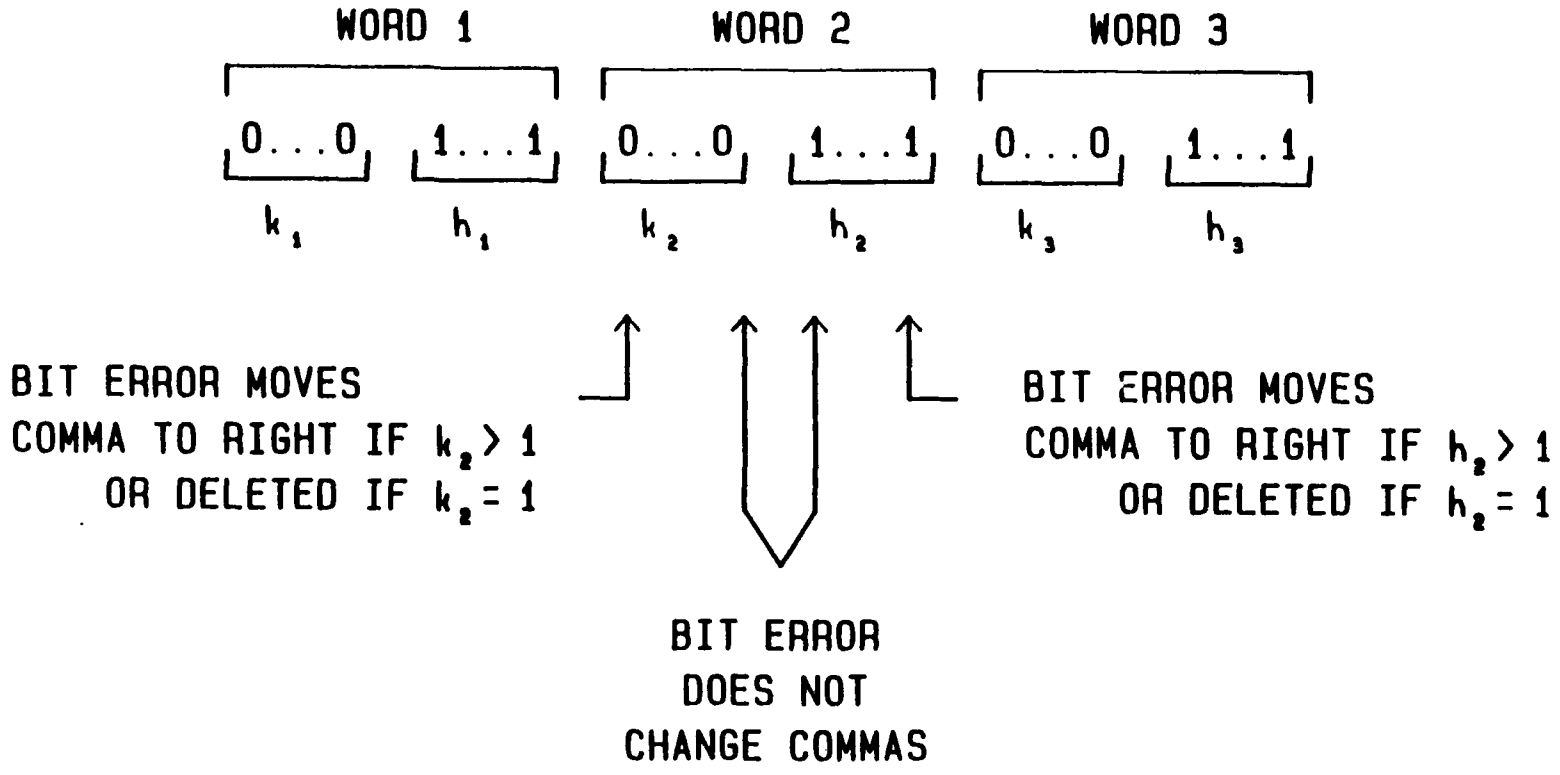


Figure 9. Impact of Bit Errors on Comma Placement for Suffix-Prefix Comma-Free Code

prefix code because the suffix-suffix code words have a more complicated structure.

Figure 10 illustrates the impact of errors for different bit positions in the middle code word of three successive code words.

The impact of an error in the first bit of the middle word depends on the first word, namely on whether or not $h1 > 0$. If $h1 > 0$ then the middle word "steals a 0 " from the first word; if $h1 = 0$ then the comma between the first and middle word is deleted and the second word is "added onto" the first word.

A bit error in the second bit position of the middle word leads one or more "0"s being added to the first word depending on whether or not $k2 > 1$.

Errors at other bit positions lead to similar behavior for the suffix-suffix code as described for the suffix-prefix code.

Our brief discussion of the suffix-suffix code indicates a clear performance difference between the suffix-prefix code and the suffix-suffix code in an error channel. Both codes share the property that a single bit error leads to at most two successive characters being decoded in error and the misplacement of at most one comma.

DATA COMPRESSION

This section contains four subsections. In the first subsection, the probabilities of occurrence of the characters appearing in four different narrative files are discussed. In the next three sections, the generalized Baudot codes, Huffman codes, and comma-free codes providing the best data compression for character encoding based on these probabilities of occurrence are identified.

Introduction

Four narrative files resident on the hard disk of the IBM PC were used to investigate compression and error properties of data compression codes. The data compression possible using character encoding is determined by the probabilities of occurrence of the characters in the data being encoded. In this section, data compression results are obtained for encoding based on character probabilities of occurrence in the four narrative files.

Table 2 summarizes the general properties of the four narrative files used throughout this study. Table 2 characterizes the four narratives in terms of the number of lines of text and the number of bytes in them. The four narratives were all technical documents involving some equations.

Table 3 completes the description of the narratives relevant to their use for data compression investigations by presenting the probabilities of occurrence for the different characters for

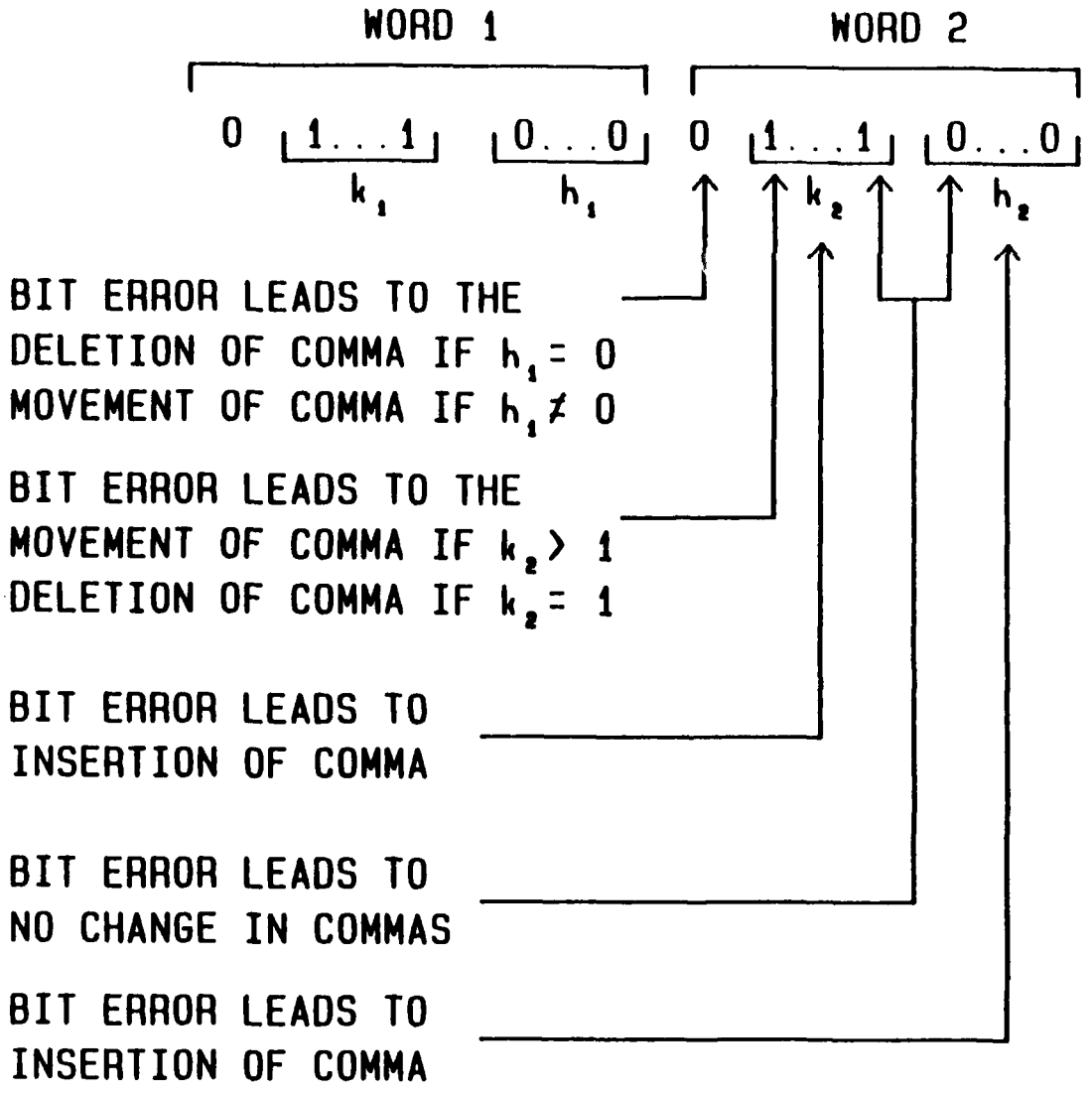


Figure 10. Impact of Bit Errors on Comma Placement for Suffix-Suffix Comma-Free Code

TABLE 2. SUMMARY OF NARRATIVE FILES
(IBM PC MASS-11 FILES)

NARRATIVE FILE	DESCRIPTION	NUMBER OF BYTES	NUMBER OF LINES
I	INVENTION DISCLOSURE OF NESTED SPATIAL-TEMPORAL INTERFERER SUPPRESSOR	31,744	612
II	INVENTION DISCLOSURE OF SPATIAL COMBINER	28,672	503
III	MEMO RE:MEECN MTG OF 19-20 NOV 1985	12,288	207
IV	ADAPTIVE ALGORITHM PERFORMANCE	13,312	290

TABLE 3. CHARACTER PROBABILITIES OF OCCURRENCE FOR FOUR NARRATIVE FILES

CHARACTER	NARRATIVE FILE I	NARRATIVE FILE II	NARRATIVE FILE III	NARRATIVE FILE IV
" "	0.3085	0.3171	0.2851	0.4095
E	0.0855	0.0865	0.0882	0.0664
T	0.0636	0.0677	0.0585	0.0492
N	0.0543	0.0537	0.0483	0.0407
O	0.0416	0.0518	0.0505	0.0420
I	0.0513	0.0511	0.0537	0.0426
A	0.0455	0.0450	0.0499	0.0434
R	0.0392	0.0421	0.0471	0.0438
S	0.0372	0.0391	0.0440	0.0367
H	0.0277	0.0263	0.0291	0.0162
C	0.0193	0.0232	0.0245	0.0216
L	0.0233	0.0218	0.0262	0.0226
D	0.0191	0.0206	0.0327	0.0165
U	0.0142	0.0185	0.0147	0.0165
P	0.0163	0.0170	0.0171	0.0164
M	0.0120	0.0162	0.0230	0.0150
F	0.0151	0.0141	0.0190	0.0169
G	0.0117	0.0121	0.0163	0.0109
B	0.0049	0.0101	0.0017	0.0067
V	0.0136	0.0099	0.0078	0.0077
W	0.0147	0.0082	0.0080	0.0030
.	0.0073	0.0076	0.0075	0.0087
Y	0.0050	0.0058	0.0074	0.0026
,	0.0062	0.0001	0.0036	0.0060
)	0.0016	0.0039	0.0024	0.0024
(	0.0016	0.0039	0.0024	0.0024
-	0.0047	0.0034	0.0044	0.0053
1	0.0078	0.0024	0.0015	0.0054
K	0.0040	0.0023	0.0014	0.0008
/	0.0003	0.0016	0.0003	0.0019

NOTE: " " denotes blank

TABLE 3. CHARACTER PROBABILITIES OF OCCURRENCE FOR FOUR NARRATIVE FILES (CONT.)

CHARACTER	NARRATIVE FILE I	NARRATIVE FILE II	NARRATIVE FILE III	NARRATIVE FILE IV
J	0.0016	0.0014	0.0011	0.0000
X	0.0018	0.0013	0.0017	0.0008
2	0.0059	0.0013	0.0016	0.0031
+	0.0031	0.0011	0.0000	0.0000
=	0.0019	0.0010	0.0002	0.0000
Z	0.0002	0.0009	0.0011	0.0004
Q	0.0014	0.0008	0.0012	0.0004
3	0.0015	0.0007	0.0010	0.0018
0	0.0009	0.0005	0.0014	0.0050
~	0.0000	0.0005	0.0000	0.0000
"	0.0003	0.0003	0.0003	0.0000
4	0.0000	0.0003	0.0002	0.0001
^	0.0014	0.0002	0.0000	0.0000
:	0.0006	0.0002	0.0008	0.0004
8	0.0001	0.0002	0.0005	0.0015
@	0.0000	0.0002	0.0000	0.0000
5	0.0001	0.0002	0.0005	0.0014
9	0.0000	0.0002	0.0007	0.0034
*	0.0021	0.0002	0.0000	0.0000
;	0.0000	0.0002	0.0006	0.0000
6	0.0000	0.0001	0.0002	0.0015
>	0.0000	0.0001	0.0001	0.0000
7	0.0000	0.0001	0.0001	0.0001
'	0.0000	0.0001	0.0002	0.0004
<	0.0000	0.0000	0.0002	0.0000
!	0.0000	0.0000	0.0001	0.0000
#	0.0018	0.0000	0.0000	0.0000
%	0.0000	0.0000	0.0001	0.0000

NOTE: " " denotes blank

each of the four narrative files. The character ordering is based on the probabilities of occurrence of the characters. The probabilities of occurrence of the characters are similar for the four narrative files. This similarity becomes clearer when cumulative probabilities of occurrence are examined for the characters partitioned in subsets of 15 characters. Table 4 presents the sums of the probabilities of occurrence for the characters in nominal 15 character subsets.

In the next subsection, we show how the probabilities in table 3 can be exploited through the design of codes with a block-like structure (generalized Baudot codes). In the next subsection, the average number of bits-per-character is calculated for the Huffman codes constructed using these probabilities of occurrence. In the third subsection, a technique is presented for matching as closely as possible the distribution of available word lengths for comma-free codes to those provided by a Huffman code.

Table 3 suggests that if we could match the code word lengths of the words assigned to the 10 to 15 characters with the highest probabilities of occurrence, we should achieve nearly the same compression for a comma-free code as for a Huffman code. This turned out to be the case and motivated the order chosen to present the material in this section.

Generalized Baudot Codes

The encoding of a 58-character set with a block code requires 6 bits. The standard Baudot code uses a shift character so that a structured code using 5 bits or 10 bits (in effect) can be utilized to code the 58-character set.

The shift symbol is not an information carrying character; i.e., the shift by itself transmits no information. The bits-per-character for the best single shift code is obtained as

5 + 5(probability of occurrence of any of the 27 least commonly occurring characters)

The average number of bits-per-character turn out to be 5.08, 5.06, 5.06, and 5.06, for narrative files I, II, III, and IV, respectively. Note, therefore, that the Baudot code represents a compression gain over block coding of 6/5 = 1.20. The compression gains for Huffman and comma-free codes should be relative to the Baudot code (not a block code). For this reason, the compression gains implied by the results obtained using these codes in this study are less than those generally quoted in the literature on these codes. In the literature, the block code is usually taken as the basis for compression calculations.

Consider a generalized Baudot code using more than one shift symbol. Suppose, in particular, that the shifts were used to produce a code with 15 words of length 4, 15 of length 8, 15 of length 12, and 13 of length 16. One of the first 16 code words

TABLE 4. PROBABILITIES OF CHARACTER OCCURRENCE FOR ANY CHARACTER IN 15 CHARACTER SUBSETS OF THE FOUR NARRATIVE FILES

NARRATIVE FILE	CHARACTERS 1-15	CHARACTERS 16-30	CHARACTERS 31-45	CHARACTERS 46-58
I	.849	.135	.0158	.0002
II	.882	.106	.011	.001
III	.879	.108	.012	.001
IV	.884	.102	.014	.000

is a shift, i.e., leads to a different interpretation of the next code word, one of these code words is reserved to lead to still another interpretation of the next code word, and one of these is reserved to lead to still another interpretation of the next code word. In each case a shift only applies to the next code word. The number of bits-per-character for this particular code is given by:

4 + 4 (probability of occurrence characters 16-30)

+ 4 (probability of occurrence characters 31-45)

+ 4 (probability of occurrence characters 46-58)

where the characters have been successively numbered beginning with the most commonly occurring character and ending with the least commonly occurring character. The average number of bits-per-character required to transmit information using this code is 4.68, 4.52, 4.54, and 4.52, for narrative files I, II, III, and IV, respectively.

Further generalizations of Baudot codes do not seem promising. For example, an attempt to match the distribution of word lengths for Huffman codes by a code build in terms of multiples of 3 leads to the following code word structure: 6 words of length 3, 12 words of length 6, 30 words of length 9, and 10 words of length 12. The average number of bits-per-character for this code was found to be 4.5 bits-per-character for narrative file I. It provides slightly greater compression with a far greater complexity that the code based on multiples of 4.

Huffman Codes

The structure of a Huffman code in the sense of its distribution of lengths of code words is determined by the probabilities of occurrence of the 58 characters in the narrative file (provided some convention to treat equi-probable sets in the construction process is adopted). Table 5 summarizes the code words assigned by the particular computer implementation of the Huffman constructed process that we used in our study. These code word lengths were obtained using the probabilities of occurrence of the characters presented in table 3 for the four narrative files. The order of the characters is the same in table 5 as that in table 3 and the characters are partitioned into sets of 15 characters to facilitate our discussion of table 5.

Recall that the probability of occurrence of one of the first 15 characters listed in table 5 exceeds .84 for all the narrative files. The word lengths assigned to the first 15 characters based on the probabilities of occurrence of the characters in the different narrative files never differ by more than one bit. The word lengths are nearly the same for the next 15 characters and tend to differ greatly only for the least

TABLE 5. HUFFMAN CODE WORD LENGTHS FOR FOUR NARRATIVE FILES

CHAR	WORD LENGTHS FOR NARRATIVE FILE I	II	III	IV	CHAR	WORD LENGTHS FOR NARRATIVE FILE I	II	III	IV
" "	2	2	2	1	J	9	9	10	22
E	3	3	4	4	X	9	9	9	11
T	4	4	4	4	2	10	8	10	9
N	4	4	4	5	+	10	8	18	15
O	4	5	4	5	=	10	9	13	21
I	4	4	4	5	Z	10	12	10	12
A	4	4	4	5	Q	10	10	10	12
R	5	5	4	5	3	10	9	10	10
S	5	5	5	5	0	11	10	9	8
H	5	5	5	6	~	11	25	19	27
C	5	6	5	6	"	11	12	11	18
L	6	5	5	6	4	12	15	12	13
D	6	6	5	6	^	12	9	19	27
U	6	6	6	6	:	12	11	10	11
P	6	6	6	6	8	12	14	11	10
M	6	6	6	6	@	12	18	16	24
F	6	6	6	6	5	12	12	11	10
G	6	6	6	7	9	13	22	10	9
B	7	8	6	8	*	12	9	17	26
V	7	6	7	7	;	12	24	11	19
W	7	6	7	9	6	13	19	13	10
.	7	7	7	7	>	13	17	13	23
Y	7	8	7	9	7	14	20	13	14
,	7	7	8	8	'	15	21	12	11
)	8	7	9	9	<	16	25	12	20
(	8	7	9	9	!	17	23	13	17
-	8	8	8	8	#	18	13	15	25
1	9	7	9	8	%	18	16	14	16
K	9	8	9	11					
/	9	11	11	9					

NOTE: " " denotes blank

probable characters. This means that use of any of the four narrative files as a training file should lead to similar compression results for encoding the narrative files.

Some of the differences between word lengths presented in table 5 could have been lessened by adopting a different convention for equi-probability character sets. In particular, a different convention should have been adopted for treating characters with zero probability to ensure that they would all be assigned code words of the same length or nearly the same length. This was discovered after the fact. For example, a better assignment of code words to zero probability of occurrence characters could have been achieved by assigning a very small probability of occurrence, say one .0000001 to each of them.

The data compression performance can be summarized by the average number of bits-per-character required to transmit the different narrative files using the four Huffman codes associated with their differing probabilities of occurrence. We refer to the narrative file used to estimate character probabilities of occurrence as the training file.

The average number of bits-per-character required to encode the training file itself can be calculated directly as the sum of the probabilities of occurrence of a character with the length of the code word assigned to it for the training file (the value obtained by summing the entries in the second columns of tables 1 through 4).

The average number of bits-per-character required to encode the remaining three narrative files using a Huffman code constructed from the training file is obtained by multiplying each character probability of occurrence in the narrative file under consideration by the length of the Huffman code assigned to that character and summing the results.

Table 6 summarizes the results of the Huffman code average bits-per-character calculations. Note that using narrative files II and III as training files gave nearly the same results. The maximum difference between two entries of the tables occurred when narrative file IV was used as a training file for narrative file I; the difference was only .23 bits-per-character.

Most of the reduction in the average number of bits required to transmit a character shown in table 5 occurs because of the high probability of occurrence of a blank in the narrative files. This can be seen by calculating the average number of bits assigned to the non-blanks for the Huffman codes assigned to each of the narratives using it as a training file.

Let $p(x)$ denote the probability of occurrence of a blank for a narrative x and let $n(x)$ denote the code word assigned to blanks for that narrative. Then the average number of bits per non-blank character $b^$ can be calculated from the average value

TABLE 6. AVERAGE NUMBER OF BITS-PER-CHARACTER FOR HUFFMAN CODES AND DIFFERENT TRAINING FILES

TRAINING FILE	NARRATIVE FILE I	NARRATIVE FILE II	NARRATIVE FILE III	NARRATIVE FILE IV
I	4.04	4.05	4.24	3.77
II	4.12	3.95	4.19	3.73
III	4.12	3.96	4.15	3.72
IV	4.27	4.05	4.32	3.63

for all characters b (values given by the diagonal entries in the table 5) for manuscript x by using the formula:

$$b^{\wedge} = (b - p(x)n(x))/(1-p)$$

Using this formula, $b^{\wedge}$ values of 4.9, 4.9, 4.8, and 6.1 bits-per-character were obtained for narratives I, II, III, and IV, respectively.

Comma-free Codes

Surveying Comma-Free Codes

The construction of a Huffman code leads to a code providing the best data compression. The construction process described by R. A. Scholtz leads to a family of codes with word lengths depending on the choices of suffixes and prefixes used in the steps of the construction process. R. A. Scholtz does not discuss how to match the comma-free construction process to the probabilities of occurrence of the characters to be encoded to provide the best compression. We have found a solution to this problem.

In this section, we develop an approach to surveying the distributions of code word lengths that can be obtained by different choices of suffixes and prefixes. In the next subsection, we show how to choose a comma-free code that gives the best compression given a character set and the probabilities of occurrence of the characters in the set.

The method that we found allows us to survey codes in terms of the distributions of their code words and can be conducted without specifying the particular code word chosen at each step of the construction process, or whether the chosen word at each step is used as a suffix or a prefix. All that need be specified is the lengths of the words chosen for suffixes and prefixes.

The R. A. Scholtz comma-free code construction process is sequential. A natural way to survey the codes is to survey them inductively based on the construction steps. We proceed to make this idea precise.

Let C[k] denote the set of code words produced after the first k steps of the construction process. Let C[0] = { 0, 1 } be the starting point in the construction process.

We seek to describe the distribution of code words by length in the set C[k+1] given the distribution of code words by length in the set C[k] and the choice of a word of length s as either a suffix or prefix in the (k+1)-th step of the construction process of R. A. Scholtz.

Let n[k](j) denote the number of code words of length j in set C[k]. For example, suppose C[1] is constructed from C[0]

using either "0" or "1". In this case, n[0](1) = 2 and n[0](j) = 0 for j > 1 and n[1](1) = 1 for j > 0.

In the suffix/prefix construction process a word used as a suffix or prefix can no longer be used as a code word. It is convenient to describe n[k+1](j) in terms of n^[k](j) when a word of length s is chosen for constructing C[k+1] from C[k].

Let

$$n\hat{}[k](j) = n[k](j) - 1 \text{ if } j = s$$

$$= n[k](j) \text{ if } j \neq s$$

Then

$$n[k+1](j) = n\hat{}[k](j) + n\hat{}[k](j-s) + \ldots + n\hat{}[k](j-ns)$$

with the convention that n^[k](j-ns) = 0 if j - ns < 1

In words, to obtain the number of code words of length j in set C[k+1] one simply adds the numbers of code words in C[k] (excluding the single code word used as a suffix or prefix, namely, n^[k](j)) plus those which could be obtained by adding the suffix/prefix to available words of length j-s (namely, n^[k](j-s)) plus those which could be obtained by adding the suffix/prefix word twice to available words of length j-2s (namely, n^[k](j-2s)), etc.

The above procedure is ideal for compiling tabular summaries of distributions of code word lengths for available comma-free codes constructed using the suffix/prefix process. Tables 7 through 10 present word length distributions of some of the comma-free codes which can be constructed in this manner. With the exception of the first column, the numbers of code words in a code are only summarized up through the length of code word needed to allow the coding of 58 characters. There are an infinite number of code words of ever increasing lengths available for each of the codes.

The codes summarized in the tables all begin with C[0] = { 0, 1 }, C[1] constructed using a length 1 code word (by necessity), and C[2] constructed from C[1] using the other available length 1 code word (not a necessity).

The columns contain the number of code words of the length labeling the rows. The number enclosed in parentheses under the column labels is the length of the code word used to construct that code set from the code set with distribution of word lengths given by the previous column. For example, in table 7, the code set C[3] is obtained from the code set C[2] through use of a suffix or prefix word of length 2 (there happens to be only one code word of length two available in this example, it would be either "01" or "10" depending on the particular choices of prefixes or suffixes used in the construction of C[1] and C[2].

TABLE 7. PARTIAL SURVEY OF THE DISTRIBUTIONS OF CODE WORD LENGTHS FOR CODES CONSTRUCTED USING PREFIXES AND SUFFIXES OF LENGTHS 1,1,2 AND 3 IN THE FIRST THREE STAGES OF CONSTRUCTION

	COMMA-FREE CODE					
WORD LENGTH	$C_{(0)}$	$C_{(1)}^{(1)}$	$C_{(1)}^{(2)}$	$C_{(2)}^{(3)}$	$C_{(3)}^{(4)}$	$C_{(3)}^{(5)}$
1	2	1				
2		1	1			
3		1	2	2	1	
4		1	3	3	3	3
5		1	4	6	6	6
6		1	5	8	9	9
7		1	6	12	15	18
8		1	7	15	21	27
9		1	8	18	27	
10		1	9	24		
11		.	10			
12		.	11			

NOTE: TABLE ONLY INCLUDES CODE WORD LENGTHS NECESSARY TO REACH 58 OR MORE CODE WORDS

TABLE 8. PARTIAL SURVEY OF THE DISTRIBUTIONS OF CODE WORD LENGTHS FOR CODES CONSTRUCTED USING PREFIXES AND SUFFIXES OF LENGTHS 1,1, AND 3 IN THE FIRST THREE STAGES OF CONSTRUCTION

	COMMA-FREE CODE				
WORD LENGTH	$C_{(0)}$	$C_{(1)}$	$C_{(1)}$	$C_{(3)}$	$C_{(3)}$
1	2	1			
2		1	1	1	1
3		1	2	1	0
4		1	3	3	3
5		1	4	5	6
6		1	5	6	6
7		1	6	9	12
8		1	7	12	17
9		1	8	14	20
10		1	9	18	
11		.	10		
12		.	11		

NOTE: TABLE ONLY INCLUDES CODE WORD LENGTHS NECESSARY TO REACH 58 OR MORE CODE WORDS

TABLE 9. PARTIAL SURVEY OF THE DISTRIBUTIONS OF CODE WORD LENGTHS FOR CODES CONSTRUCTED USING PREFIXES AND SUFFIXES OF LENGTHS 1,1,2 AND 4 IN THE FIRST THREE STAGES OF CONSTRUCTION

	COMMA-FREE CODE					
WORD LENGTH	$C_{(0)}$	$C_{(1)}$	$C_{(1)}$	$C_{(2)}$	$C_{(4)}$	$C_{(4)}$
1	2	1				
2		1	1			
3		1	2	2	2	2
4		1	3	3	2	1
5		1	4	6	6	6
6		1	5	8	8	8
7		1	6	12	14	14
8		1	7	15	17	18
9		1	8	18	24	30
10		1	9	24		
11		1	10			
12		1	11			

NOTE: TABLE ONLY INCLUDES CODE WORD LENGTHS NECESSARY TO REACH 58 OR MORE CODE WORDS

TABLE 10. PARTIAL SURVEY OF THE DISTRIBUTIONS OF CODE WORD LENGTHS FOR CODES CONSTRUCTED USING PREFIXES AND SUFFIXES OF LENGTHS 1,1, AND 4 IN THE FIRST THREE STAGES OF CONSTRUCTION

	COMMA-FREE CODE			
WORD LENGTH	$C_{(0)}$	$C_{(1)}$	$C_{(2)}$	$C_{(3)}$
		$C_{(1)}$	$C_{(1)}$	$C_{(4)}$
1	2	1		
2		1	1	1
3		1	2	2
4		1	3	2
5		1	4	4
6		1	5	6
7		1	6	8
8		1	7	9
9		1	8	12
10		1	9	15
11		1	10	22
12		1	11	

NOTE: TABLE ONLY INCLUDES CODE WORD LENGTHS NECESSARY TO REACH 58 OR MORE CODE WORDS

Figures 11 and 12 illustrate the tailoring of the distribution of word lengths possible by using some of the two- or three-step constructions and some of the two- or four-step constructions, respectively. The figures are constructed to compare the code words available through selected comma-free constructions to encode a 58-character set. Each curve is labeled by the suffix/prefix word lengths leading to the plotted code set.

Choice of a Comma-free Code for a 58-character Set

It is possible to calculate the average number of bits-per-character for a given character set and the probabilities of occurrence of the characters in the sets for each candidate comma-free code. The calculation would consist of first ordering the characters by probability of occurrence (say from highest to lowest) and assigning code words to the characters by ordering the available code words from shortest to longest. A sum over the character set of the probability of occurrence of a character multiplied by the character code word length then is the average number of bits-per-character for the code.

It is possible to reduce the comma-free code candidates to a few obvious front-runners by using Huffman code words as a gauge for the candidates.

The probabilities of occurrence for the character set used for our 58 character simulations of the Huffman code, has the property that the character probabilities fall off rapidly from the most used characters to the least used characters as shown in table 3 which was presented earlier. In such a situation, if we could closely match the code word lengths provided by the Huffman code constructed for the given character probabilities of occurrence for the first 10 to 15 characters, we would expect very similar compression performance from that comma-free and a Huffman code.

Table 11 shows how closely the simplest suffix/prefix candidate code word lengths match those provided by the Huffman code. The first four columns of word lengths repeat the information presented earlier in table 4 for Huffman codes and a fifth column presents the word lengths for any of the suffix-prefix comma-free codes obtained by use of "0" and "1" as suffixes or prefixes in a two-step construction. The assignment of code words to characters is presented for the ordering provided by the probabilities of occurrence of characters in narrative file II. However, as can be seen from table 11, this assignment leads to excellent word length agreement through the first 40 characters, regardless of which narrative file is used as the training file.

Consider the first fifteen rows of table 11 for word lengths of the Huffman code for narrative file II and for the comma-free code. For the third character ("T"), the comma-free code is one

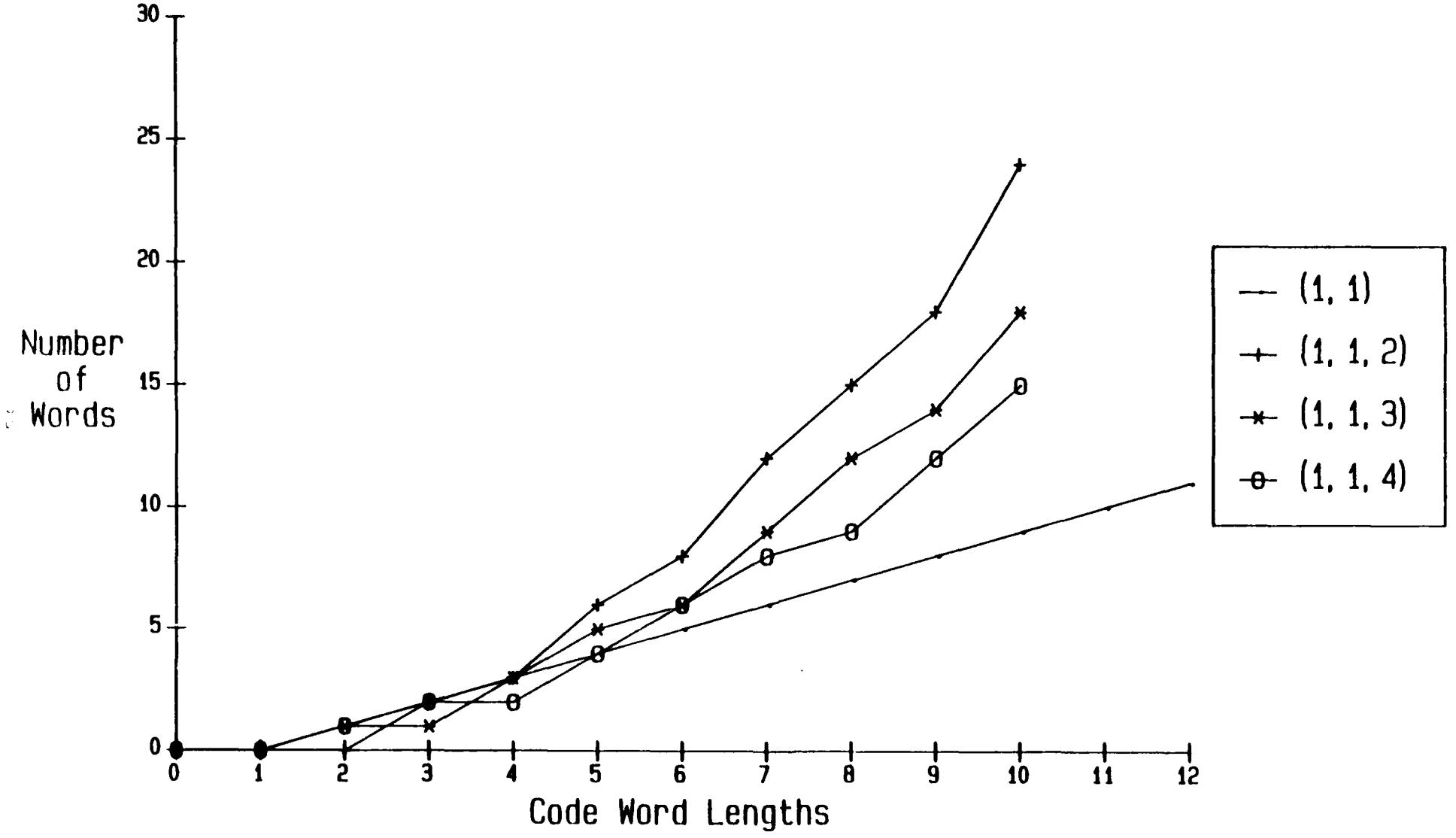


Figure 11. Word Lengths for Selected Two- and Three Step Suffix/Prefix Comma-Free Codes

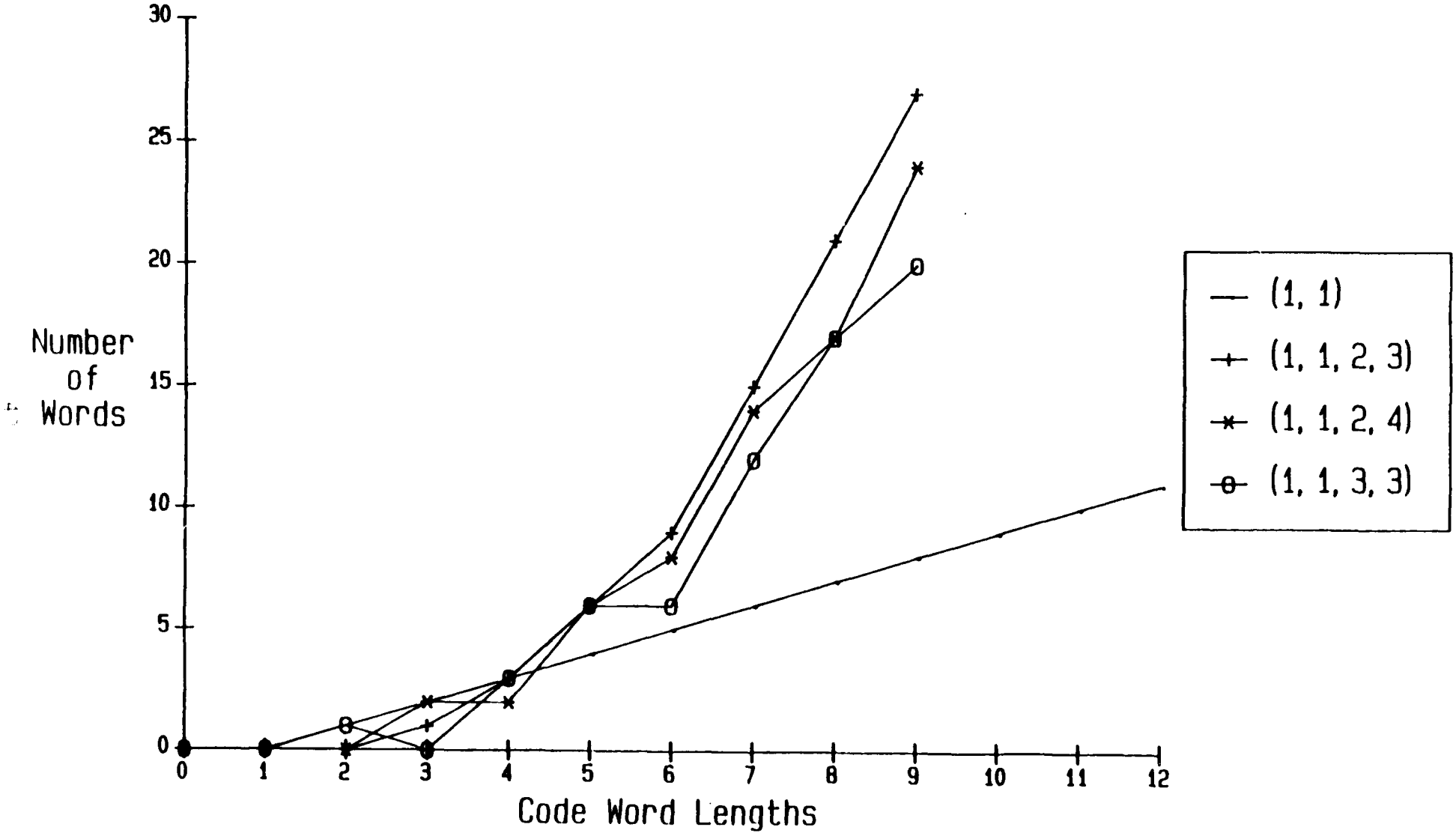


Figure 12. Word Lengths for Selected Two- and Four-Step Suffix/Prefix Comma-Free Codes

TABLE 11. HUFFMAN AND COMMA-FREE CODE WORD LENGTHS FOR FOUR NARRATIVE FILES

CHAR	WORD LENGTHS FOR NARRATIVE FILE I	II	III	IV	COMMA-FREE	CHAR	WORD LENGTHS FOR NARRATIVE FILE I	II	III	IV	COMMA-FREE
" "	2	2	2	1	2	J	9	9	10	22	9
E	3	3	4	4	3	X	9	9	9	11	9
T	4	4	4	4	3	2	10	8	10	9	9
N	4	4	4	5	4	+	10	8	18	15	9
O	4	5	4	5	4	=	10	9	13	21	9
I	4	4	4	5	4	Z	10	12	10	12	9
A	4	4	4	5	5	Q	10	10	10	12	10
R	5	5	4	5	5	3	10	9	10	10	10
S	5	5	5	5	5	0	11	10	9	8	10
H	5	5	5	6	5	~	11	25	19	27	10
C	5	6	5	6	6	"	11	12	11	18	10
L	6	5	5	6	6	4	12	15	12	13	10
D	6	6	5	6	6	^	12	9	19	27	10
U	6	6	6	6	6	:	12	11	10	11	10
P	6	6	6	6	6	8	12	14	11	10	10
M	6	6	6	6	7	@	12	18	16	24	11
F	6	6	6	6	7	5	12	12	11	10	11
G	6	6	6	7	7	9	13	22	10	9	11
B	7	8	6	8	7	*	12	9	17	26	11
V	7	6	7	7	7	;	12	24	11	19	11
W	7	6	7	9	7	6	13	19	13	10	11
.	7	7	7	7	8	>	13	17	13	23	11
Y	7	8	7	9	8	7	14	20	13	14	11
,	7	7	8	8	8	'	15	21	12	11	11
)	8	7	9	9	8	<	16	25	12	20	11
(	8	7	9	9	8	!	17	23	13	17	12
-	8	8	8	8	8	#	18	13	15	25	12
1	9	7	9	8	8	%	18	16	14	16	12
K	9	8	9	11	9						
/	9	11	11	9	9						

NOTE: THE COMMA-FREE CODE HAS BEEN CHOSEN TO BEST MATCH THE WORD LENGTHS OF THE HUFFMAN CODE FOR NARRATIVE FILE II

bit shorter than the narrative file II word length, which leads to a decrease in the expected number of bits of .0677. Only two characters ("A" and "C") of the first 15 are one bit longer, which leads to an increase in the expected number of bits of .0450 and .0232. The net difference (or penalty for using the comma-free code) between the average number of bits-per-character for the first 15 characters is only .0005 bits-per-character. For the remaining rows the penalty for using the comma-free code is at most .0162 bits for any character and only above .01 for three characters. The overall penalty for using the comma-free code rather than a Huffman code for compression (based on narrative file II) is only .045 (rounded down to three places) bits-per-character.

Table 12 presents a comparison between Huffman code bits-per-character values and comma-free code bits-per-character values for the code word to character assignments shown in table 11. Slightly lower average bits-per-character values are possible for comma-free codes encoding narrative file codes I, III, and IV than those shown in table 12. However, the performance differences between the Huffman codes and the comma-free codes are so small that we did not investigate other matchings of comma-free codes to Huffman code word lengths for these cases. The Huffman code average bits-per-character values are those obtained using the assignment of code words to a narrative file based on the probabilities of occurrence of its characters.

The next best comma-free code appears to be the code (1, 1, 3) for which similar calculations revealed a penalty of .114 (rounded down to three bits) bits-per-character for using this comma-free code instead of the Huffman code for narrative file II.

It appears from this example that similar procedures would allow us to find a comma-free code giving nearly the same compression behavior as a Huffman code, provided that the probabilities of occurrence of the characters in the character set fall off in a reasonable manner from the highest probability of occurrence to the lowest.

COMPRESSION CODE PERFORMANCE IN A CHANNEL WITH ERRORS

In this subsection we estimate the performance of generalized Baudot codes, Huffman Codes, and Comma-free codes in a channel with errors. The generalized Baudot and comma-free codes could be studied analytically. Huffman codes were studied through use of simulations. Each code is evaluated by estimating the average number of characters decoded in error per bit error and the average number of characters output by the decoder in error per bit error.

TABLE 12. COMPARISON OF BITS-PER-CHARACTER VALUES OF HUFFMAN AND COMMA-FREE CODES

NARRATIVE FILE	BITS-PER-CHARACTER	
	HUFFMAN CODE	COMMA-FREE CODE
I	4.04	4.10
II	3.95	4.00
III	4.15	4.26
IV	3.63	3.81

Generalized Baudot Codes

The performance of the generalized Baudot codes is simple to evaluate because a single bit error always leads to one character being decoded in error, whether it occurs in a code word of an information carrying character or a shift character. If the bit error occurs in the code word of an information character that character is decoded in error, and if it occurs in a shift character then that character and intervening shift characters through the first information character are output as error characters.

For the single-shift 5 bit based Baudot code, the statistics are:

1 character decoded in error per bit error

1.01 to 1.02 output character in error per bit error depending on the training file

For the three-shift 4 bit based Baudot code, the statistics are:

1 character decoded in error per bit error

1.13 to 1.17 output character in error per bit error depending on the training file

Huffman Codes

Introduction

Simulation results were obtained for the 95-character symbol set associated with the IBM PC. It was intended to develop software and plan further analysis based on these results and then to apply lessons learned to the processing of a Navy message data base. The Navy message data base was not available in a timely enough manner to allow the analysis to continue without interruption so it was decided to emulate the 58-character set (Baudot) used in Navy communications by reducing the 95-character IBM PC set to 58 characters. This was accomplished by use of the all capital letter option of the operating system and by editing the documents being processed to be free of selected special symbols. This section contains three subsections: the first describes the simulation software; the second, the results obtained for a 95-character set; and the third, the results obtained for a 58-character set.

Description of Simulation Software

The original program reads a text file and counts the number of occurrences of each character; from this, a Huffman code is constructed using the construction process first described by Huffman in his original paper. This construction process has numerous degrees of freedom. In order to study the relationship

between the performance of a Huffman code in a channel with errors and the specific choices made in the Huffman construction process, a program was written that allowed the user to specify the probability that the character set with the highest probability of occurrence would be assigned a "1" at each stage of the construction process. (Even though it turned out that the probability of a "1" occurring was not a meaningful parameter, varying the probability of a "1" occurring allowed searches to be conducted for a particular Huffman code with better performance in a channel with errors than most codes which could be constructed.) The probabilities of occurrence for each character and the assigned Huffman code words for each character are written to files so that the particular code used to obtain a particular set of performance results could always be recovered established if desired.

Four basic programs were written to exercise Huffman codes: (1) a program which would encode a message file using the Huffman code, (2) a program to introduce random bit errors into the encoded file (the probability of a bit error is user specified), (3) a program to decode the encoded bit stream, and (4) a program which compares the decoded bit stream with the original message and accumulates various error statistics.

An additional software program was developed to evaluate the feasibility of a user correcting character errors through message context. The program was interactive and allowed the user to select any character (of 20 displayed characters) of the message for possible reinitiation of the Huffman decoding process. The interactive program would retrieve the code word of the selected character and reinitiate the Huffman coding process by altering each bit in its code word. The new characters would be displayed and after the trial decodings were completed, the user could choose the most acceptable string of characters and the software would implement the bit change in the original bit stream corresponding to the selected option. The software compares the selected character strings with the original message and records the number of errors (total) and number of corrected errors.

Simulation Results for a 95-character Set

Figure 13 summarizes the results of an extensive search for the Huffman code providing the best performance in a channel with errors. The code word lengths of all of the codes constructed are the same because the grouping of the characters, as discussed earlier, is the same for all of the codes. They differ only in the choices of "1" or "0" at each stage of the Huffman construction process. The experiments were run by varying the probability that the set of characters with highest probability of occurrence at each stage was assigned a "1". The vertical scale is the average number of characters decoded in error (input characters to the decoder) per bit error. Each Huffman code was exercised against each of the four narrative files described in the previous paragraph and the maximum and minimum average number

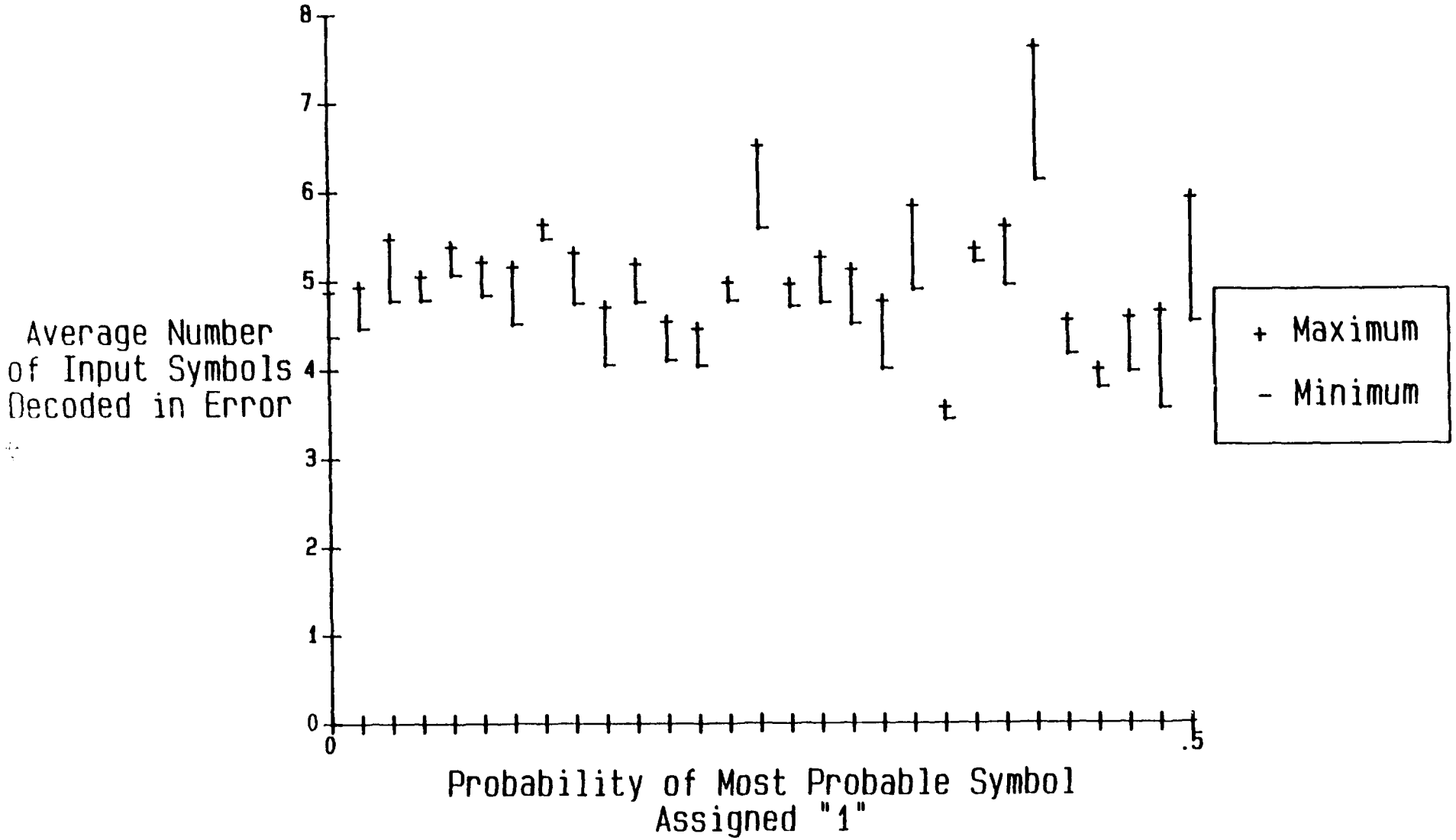


Figure 13. Decoding Error Statistics Resulting from Randomly Induced Bit Errors for a Variety of Huffman Codes (95-Character Set)

of characters decoded in error per bit error for these four narrative files plotted.

Figure 13 shows that the average number of decoded characters per bit error is definitely dependent on the Huffman code. The dependence on narrative file is small compared with the dependence upon the code used. Figure 14 shows this in another way, showing the dependence of the average number of decoded characters per bit error for the best performing code, the worst performing codes, and two selected average performing codes.

Figure 15 presents a distribution of all of the average number of decoded character per bit error values obtained for the different experiments on the narrative files presented in figure 13. It is particularly noteworthy that the four results obtained for the code with the lowest value fall into the two lowest value bins of figure 13. Figures 13 and 15 clearly show that the code with the best error ratios was clearly the best performing code in a channel with errors.

Some experiments were run using an operator-interactive program. These experiments were designed to determine the percentage of errors introduced into a text file through Huffman encoding, transmission in a noisy channel, and Huffman decoding that could be corrected through narrative context. In particular, it appears possible to use a standard spell check program as a basis for reinitializing Huffman decoding after changing a bit likely to be in error. The potential of such an algorithm could be assessed by using an operator-interactive program--with the operator choosing the decoding which provided text which made the most sense.

A 4271 character narrative file consisting of 88 lines and 4456 bytes was chosen to assess operator-interactive correcting of narrative character errors. Bit errors were introduced randomly at a rate of .005. This error rate would lead to an estimated 90 characters containing a bit error ((.005)(4271 characters)(4 bits/character)). These 90 character errors led to 362 character decoding errors. After the inter-active session the operator was able to reduce the number of character decoding errors to 85 errors (that is the number of character errors were reduced by 76 percent).

Simulation Results for a 58-character Set

A series of simulations was run to find the best performing Huffman code in a channel with errors. Trials were run using the probabilities of occurrence of the characters in each of the four narrative files. Each Huffman code was then exercised for the four narrative files, including the one from which the probabilities of occurrence of the characters were derived.

Simulations were run by randomly introducing bit errors at a rate of 3 per 1000 bits. Successive bit errors were independent

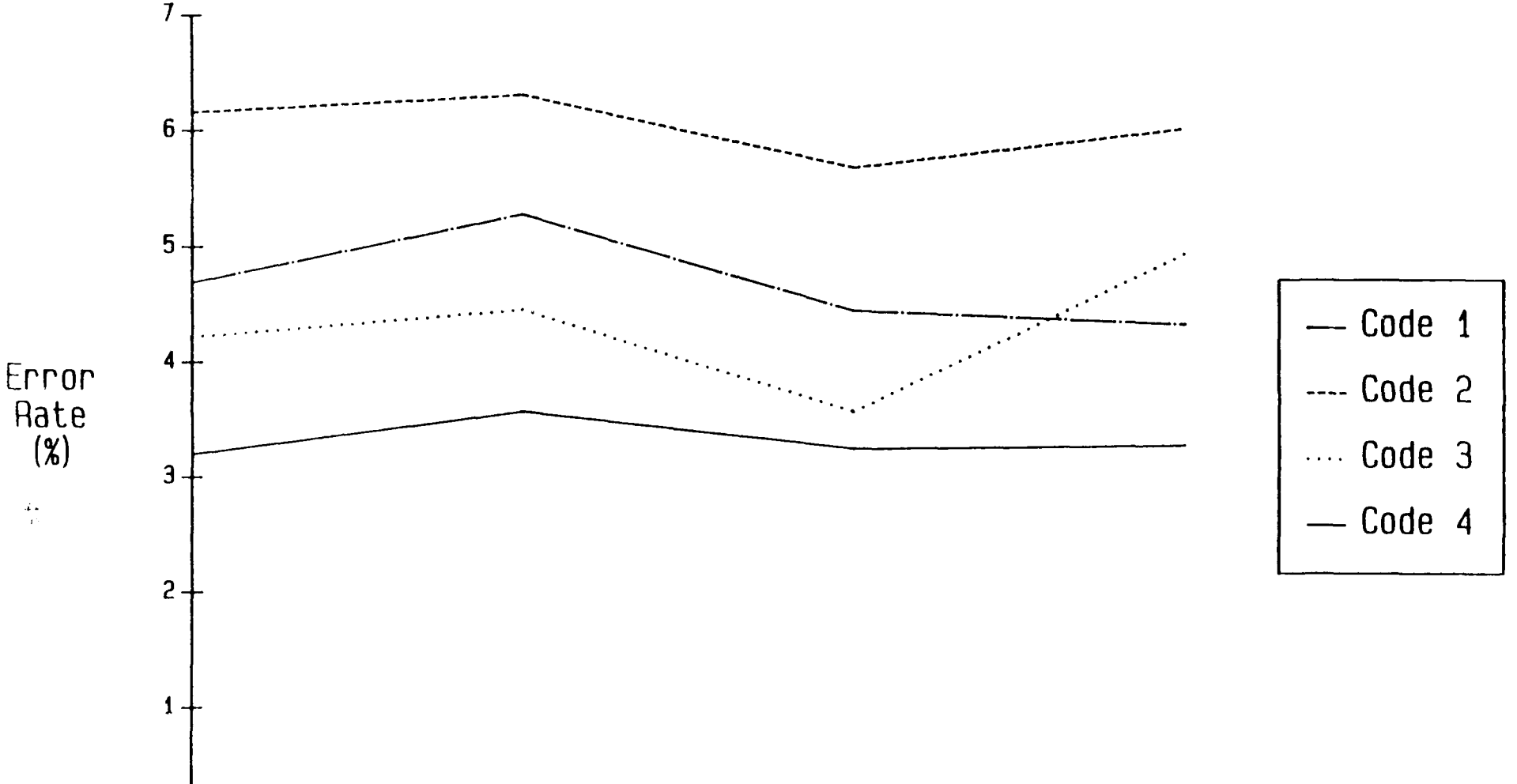


Figure 14. Average Number of Decoded Characters per Bit Error for Four Huffman Codes

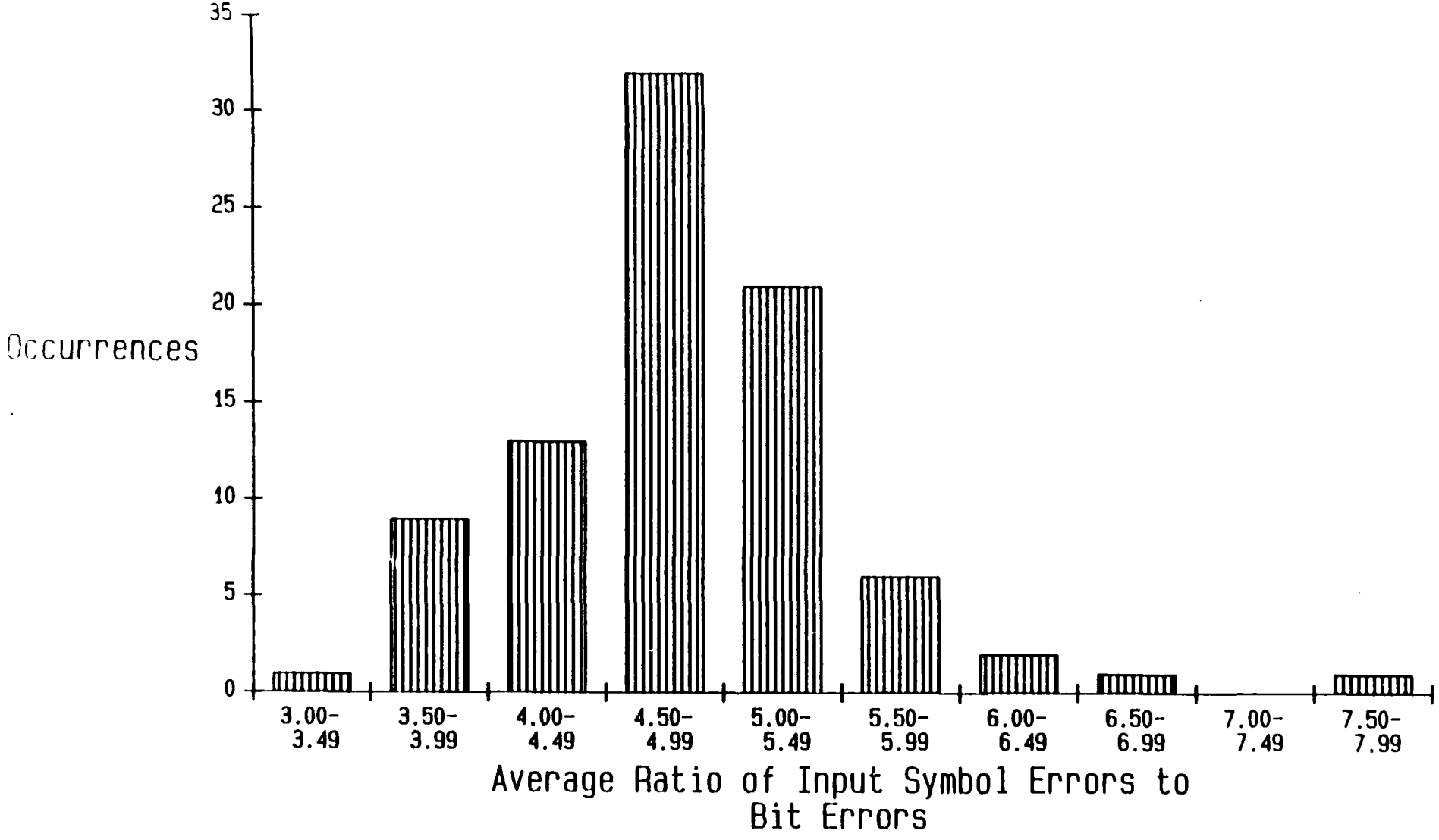


Figure 15. Distribution of the Average Ratio of Input Symbol Errors to Bit Errors for a Large Number of Equally Efficient Huffman Codes

of one another. No attempt was made to model the impact of burst errors on the channel. It was felt that should burst errors pose a problem in the implementation of a particular code, it would always be possible to superimpose interleaving after data compression encoding and deinterleaving prior to data compression decoding.

Figures 16 through 19 present the average numbers of characters decoded in error per bit error for experiments run using narrative files I, II, III, and IV, respectively, as a training file. Each figure presents the results encoded according to the narrative file for which the decoded character errors per bit value was obtained.

The first thing to observe is that in all four figures, the poorest error performance results were obtained when processing narrative file IV (the most compressible). In contrast, the best error performance was obtained for narrative file III, which was the least compressible.

The second thing to notice is that the results obtained using narrative file IV as a training file gave the best performance in general in an error channel.

In order to more completely characterize the performance of Huffman codes in channels with errors, a worst and best case for each narrative file as a training file from the viewpoint of average number of characters decoded in error per bit error were selected for further analysis. Figures 20 through 27 present the distributions of lengths of successive output characters in error obtained for the trials given for the selected simulations. Note that any of the output character error sequences may involve more than one bit error. This is likely for very long sequences of output character errors and less likely for shorter sequences because the bit errors were randomly introduced at a rate of 3 in 1000. However, the likelihood of two bit error induced error sequences merging is very small and can be neglected; therefore, output character performance is summarized in terms of the average number of output character errors per bit error.

Some of the distributions had very long sequences of errors (a maximum of 104 for narrative I as a training file), so that the distributions are presented for character error sequences of lengths 1, 2, ..., 9 and those of length 10 or greater. The maximum length of an error sequence is included in each figure in the upper right hand corner.

There is a dramatic difference in the structure of the distributions for each of the narrative files used as a training file for the Huffman codes found to give the best and worst performance in a channel with errors. The best distributions have a preponderance of short length sequences (lengths one, two, and three) while the worst distributions tend to be relatively flat with the occurrence of extremely long character error

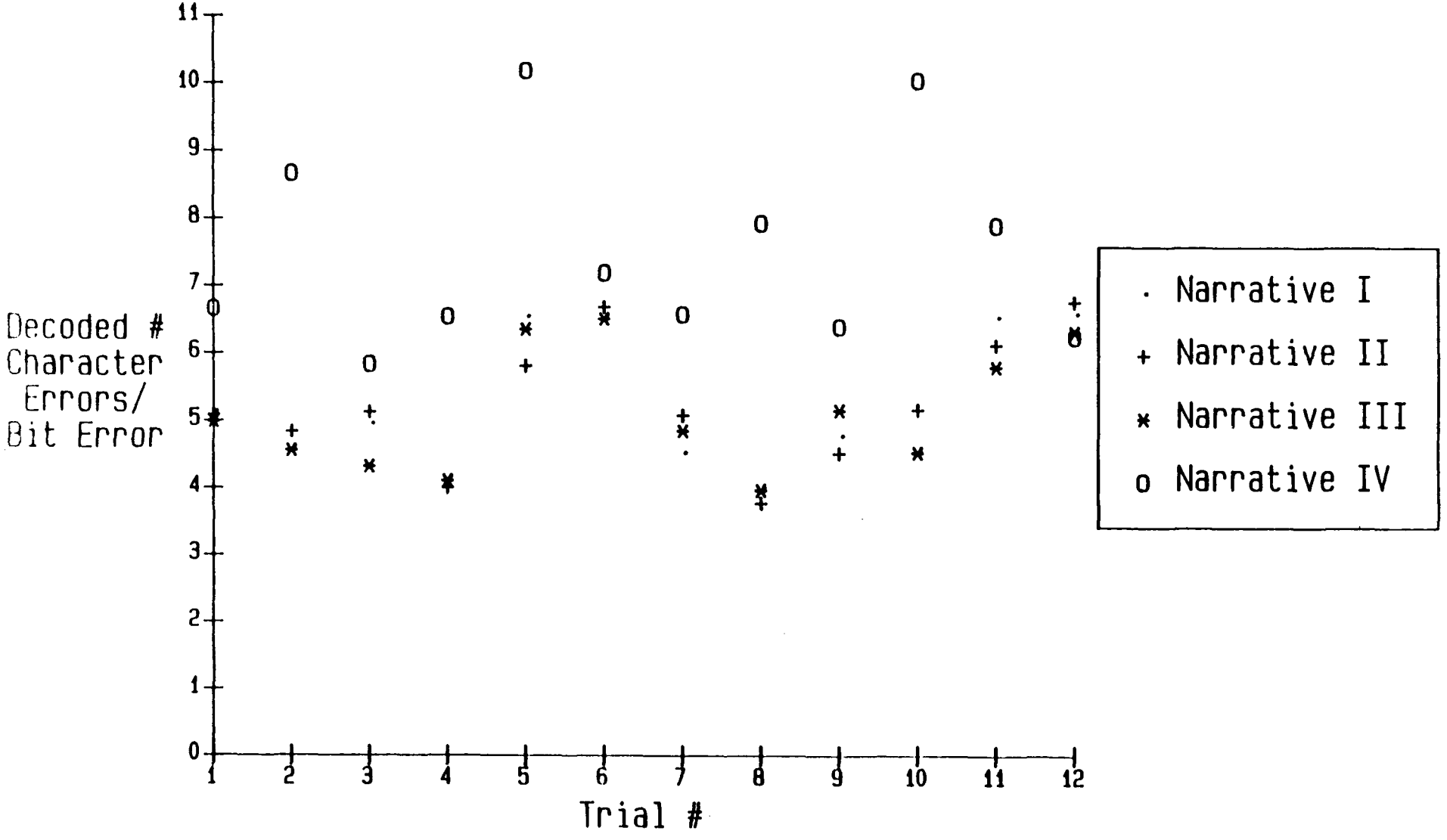


Figure 16. Character Decoding Errors for Experiments Using Narrative I as a Training File

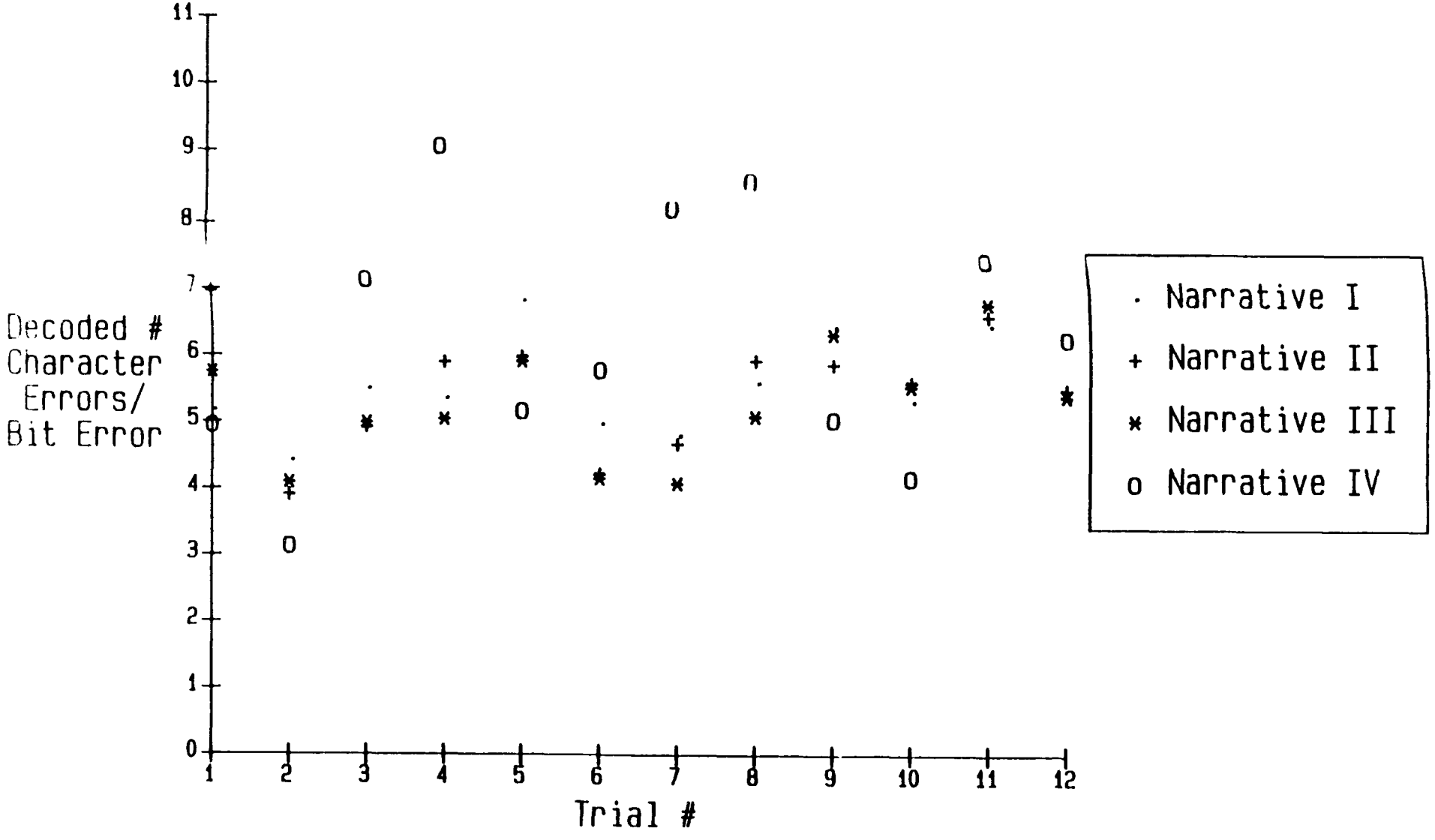


Figure 17. Character Decoding Errors for Experiments Using Narrative II as a Training File

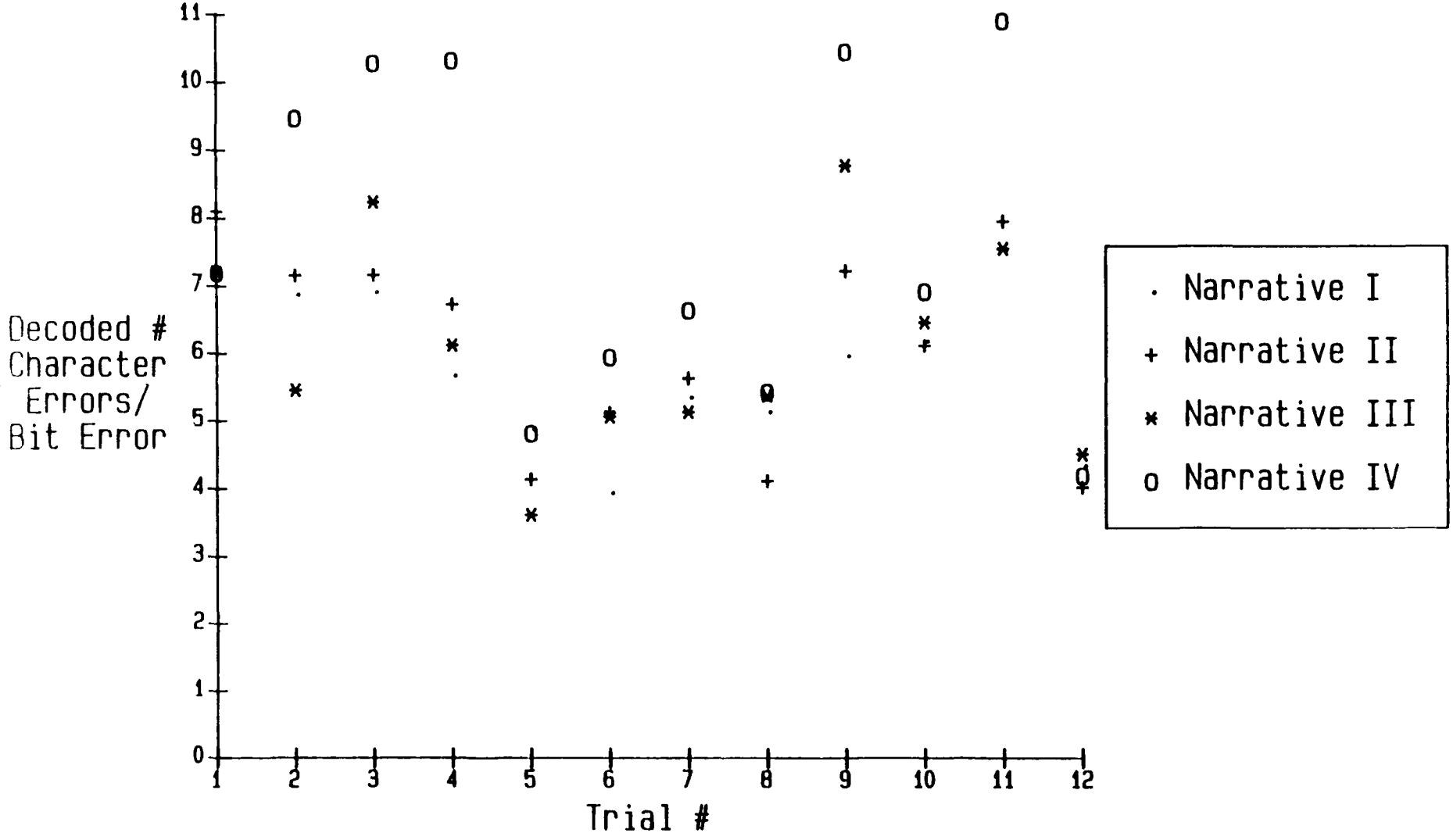


Figure 18. Character Decoding Errors for Experiments Using Narrative III as a Training File

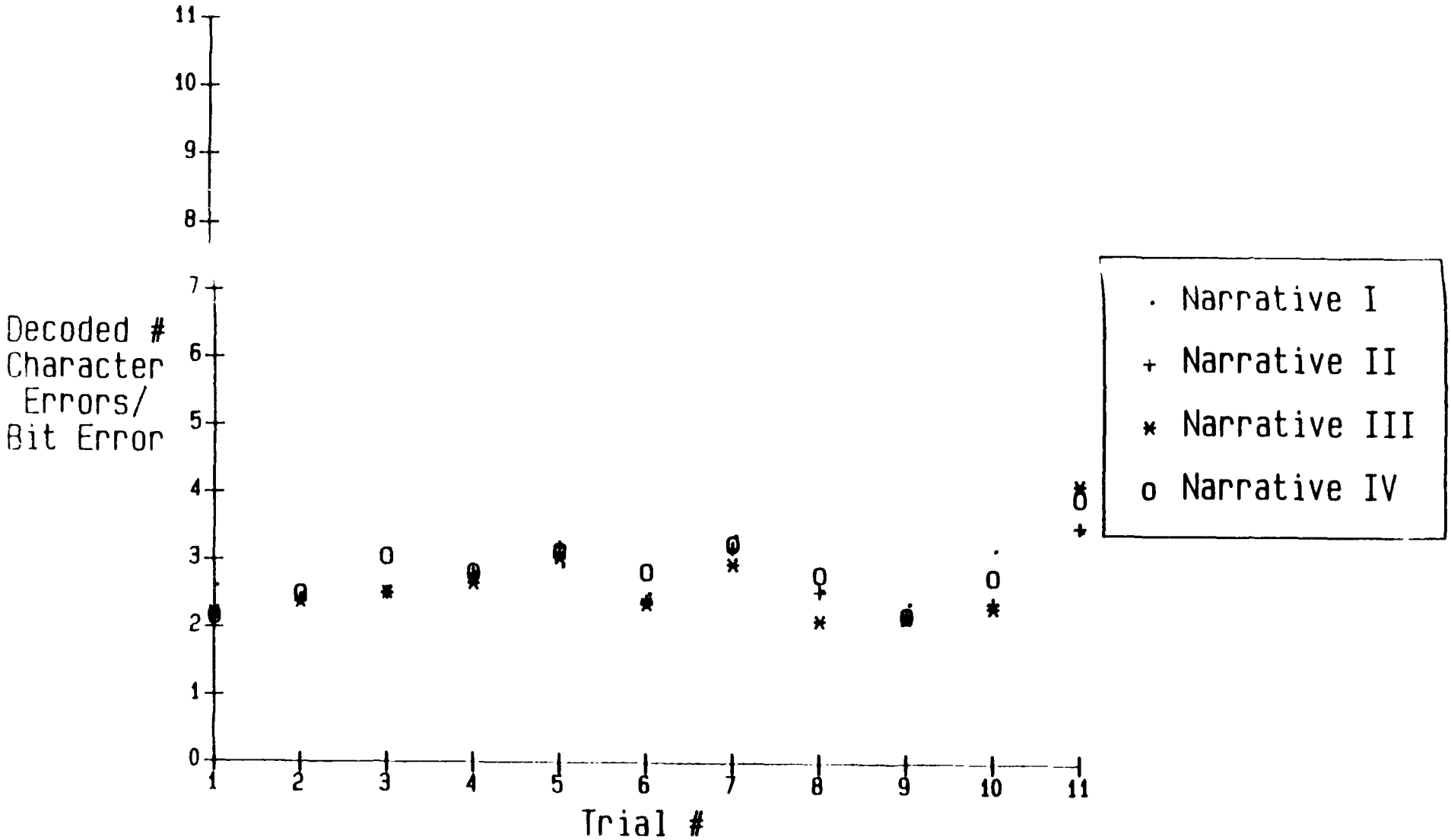


Figure 19. Character Decoding Errors for Experiments Using Narrative IV as a Training File

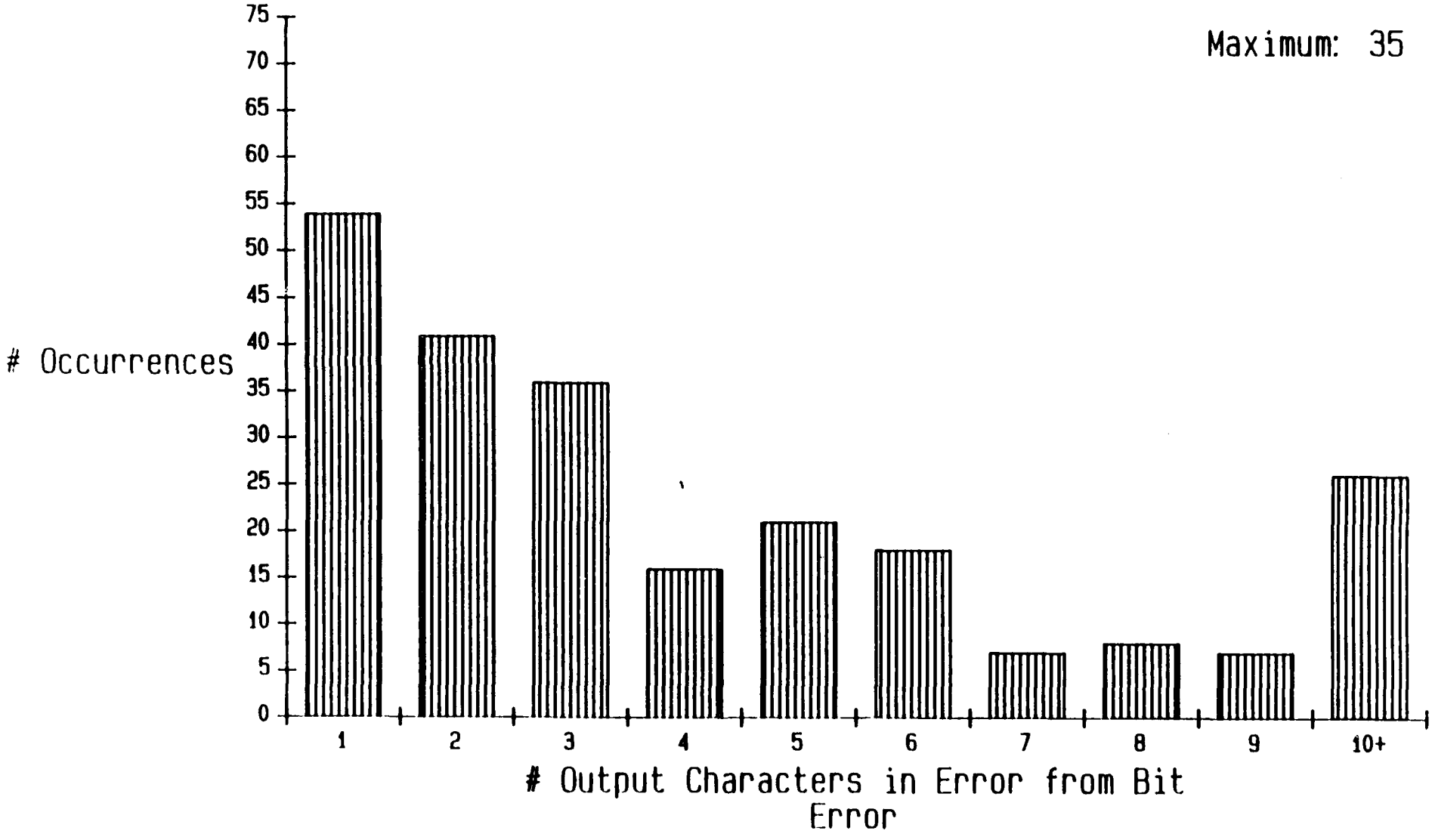


Figure 70. Distribution of Output Errors for Best 58-Character Huffman Code Using Narrative File 1 as Training File

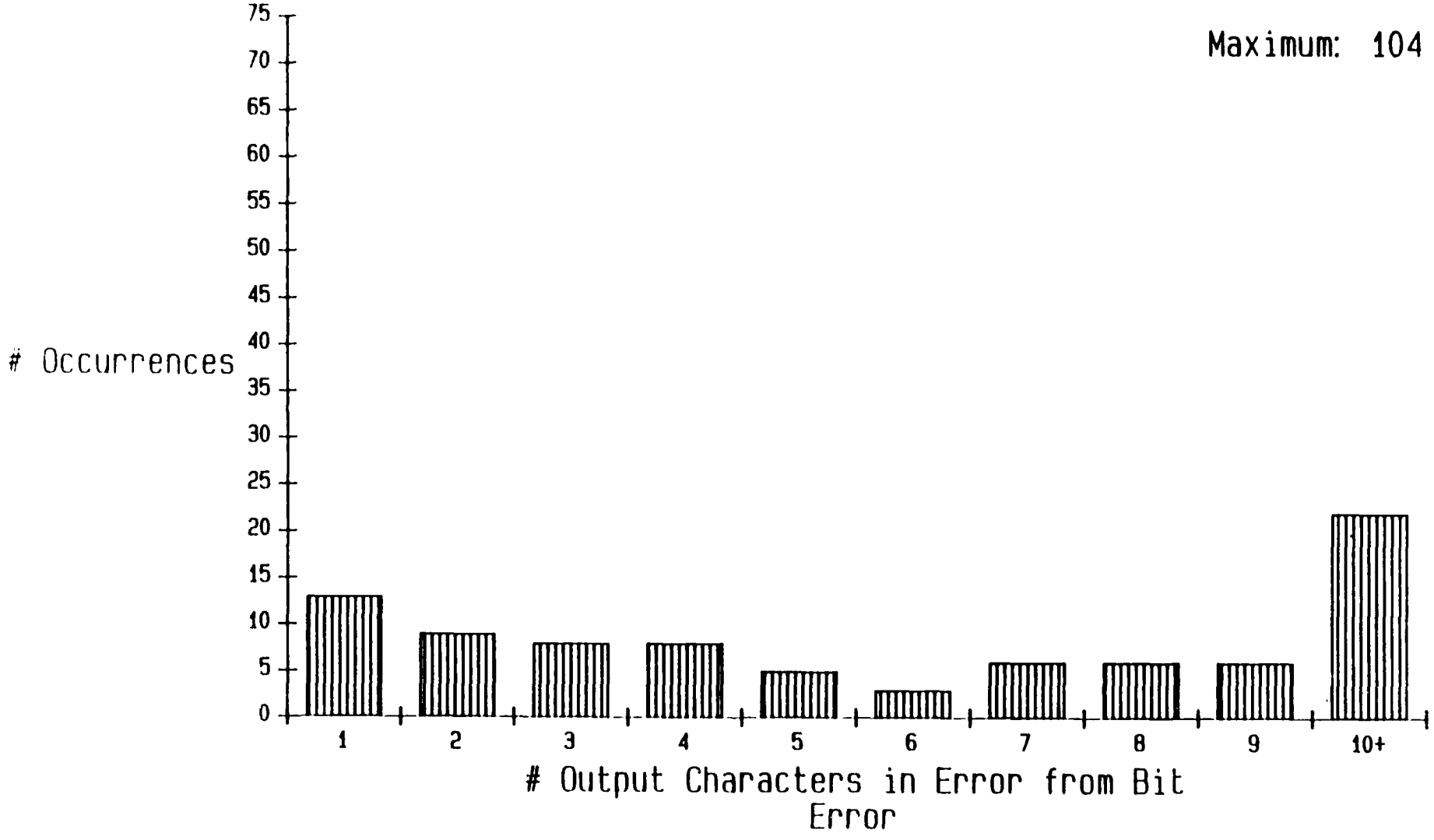


Figure 21. Distribution of Output Errors for Worst 58-Character Huffman Code Using Narrative File I as Training File

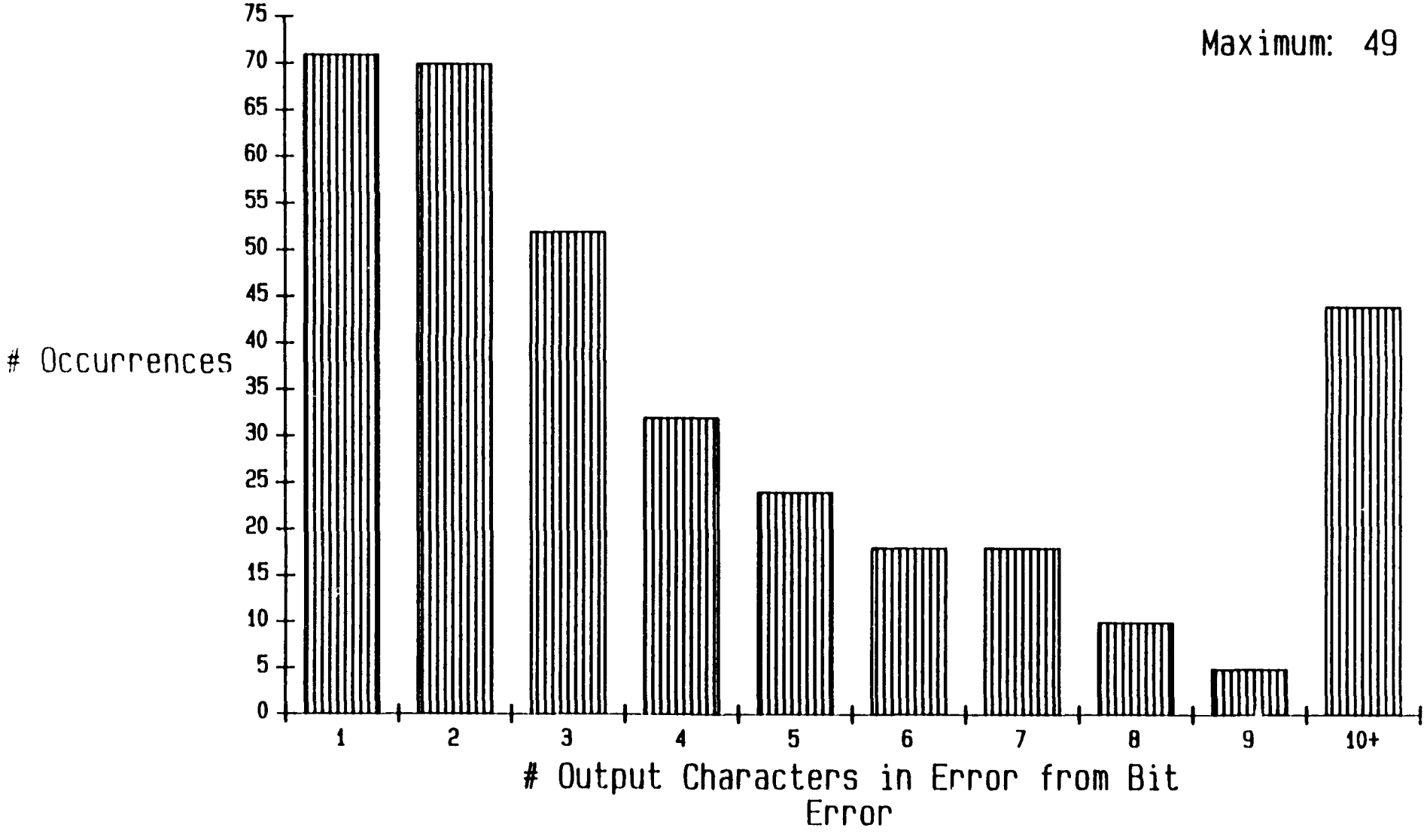


Figure 22. Distribution of Output Errors for Best 58-Character Huffman Code Using Narrative File II as Training File

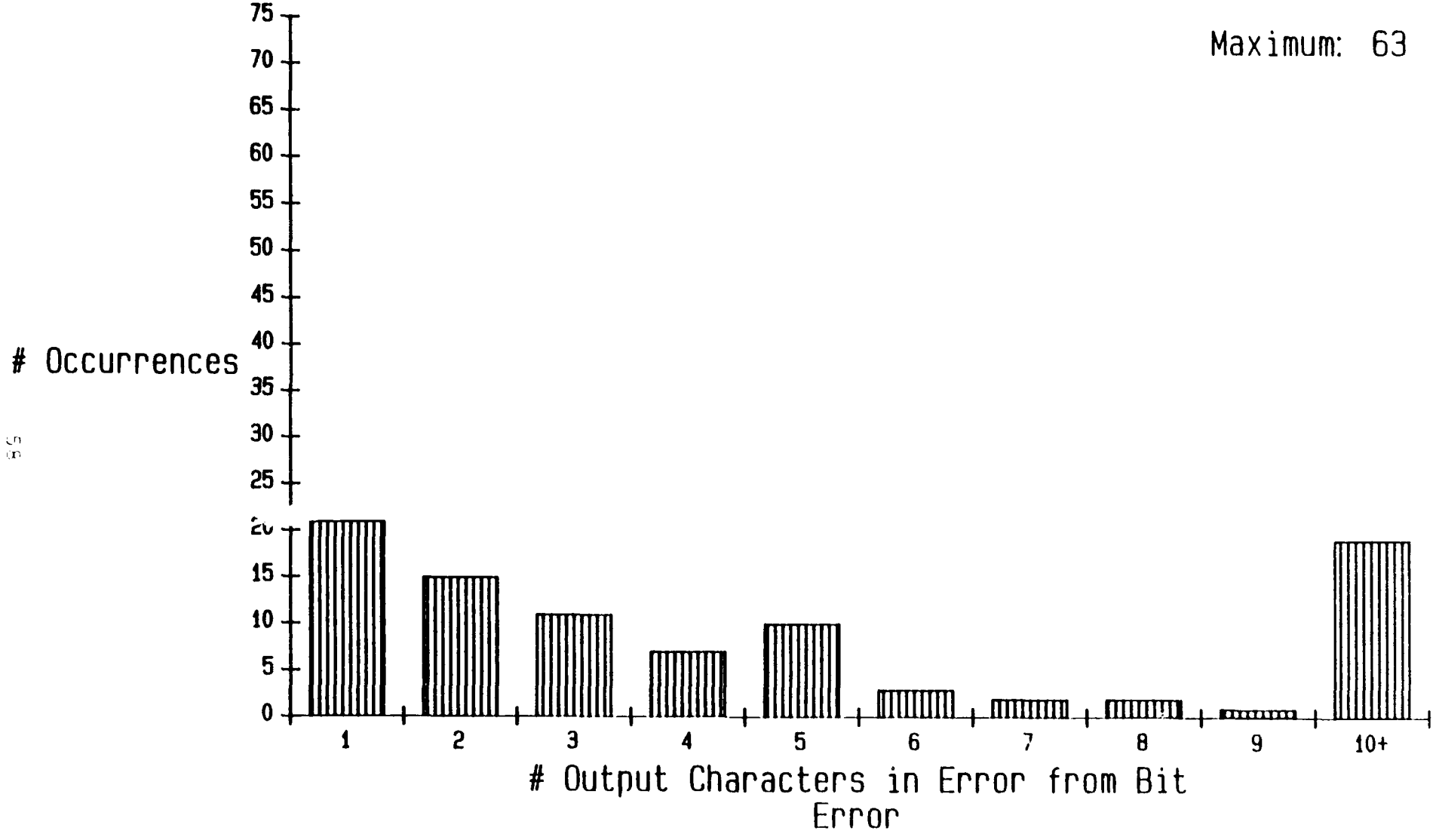


Figure 23. Distribution of Output Errors for Worst 58-Character Huffman Code Using Narrative File II as Training File

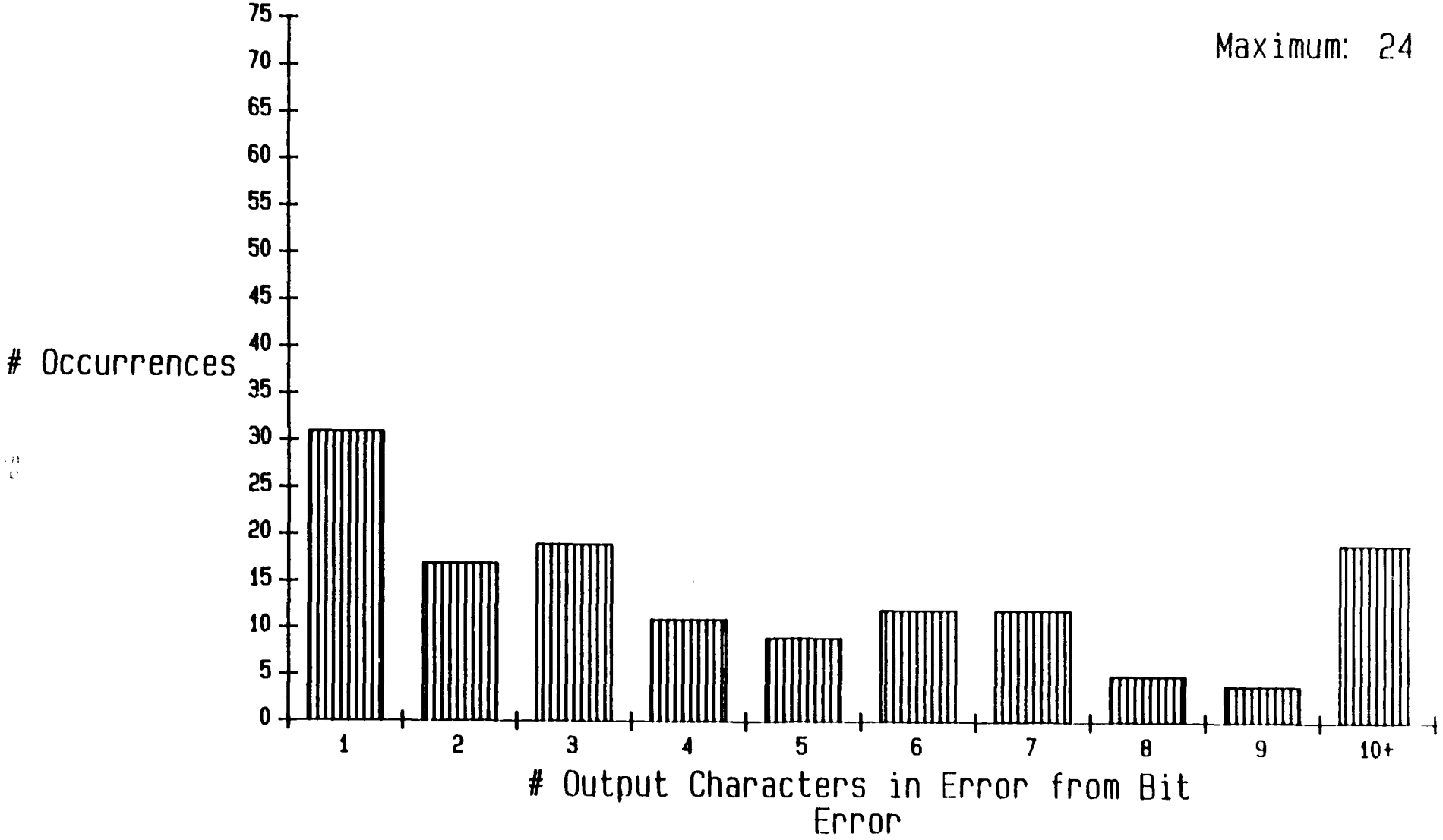


Figure 24. Distribution of Output Errors for Best 58 Character Huffman Code Using Narrative File III as Training File

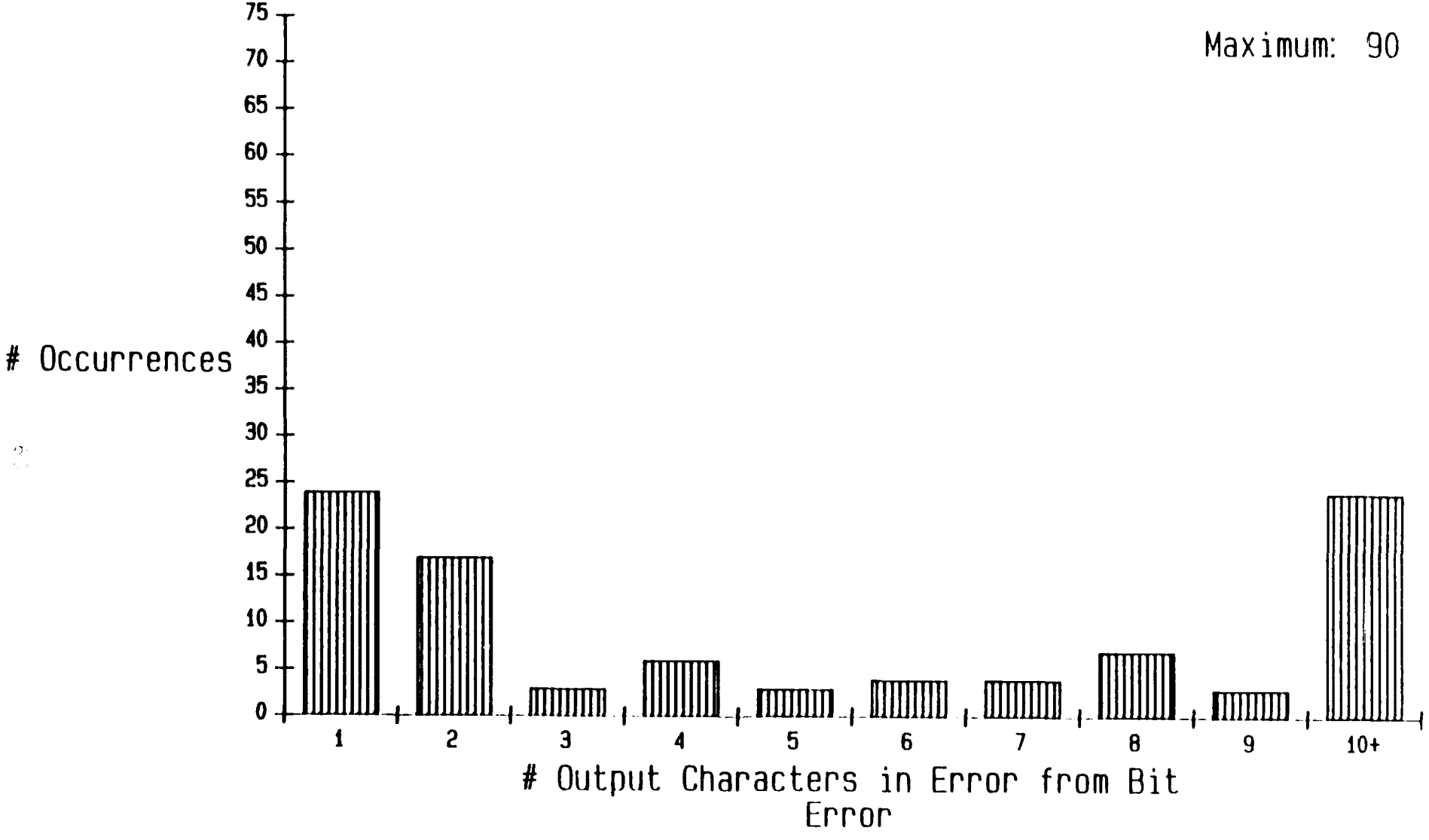


Figure 25. Distribution of Output Errors for Worst 58 Character Huffman Code Using Narrative File III as Training File

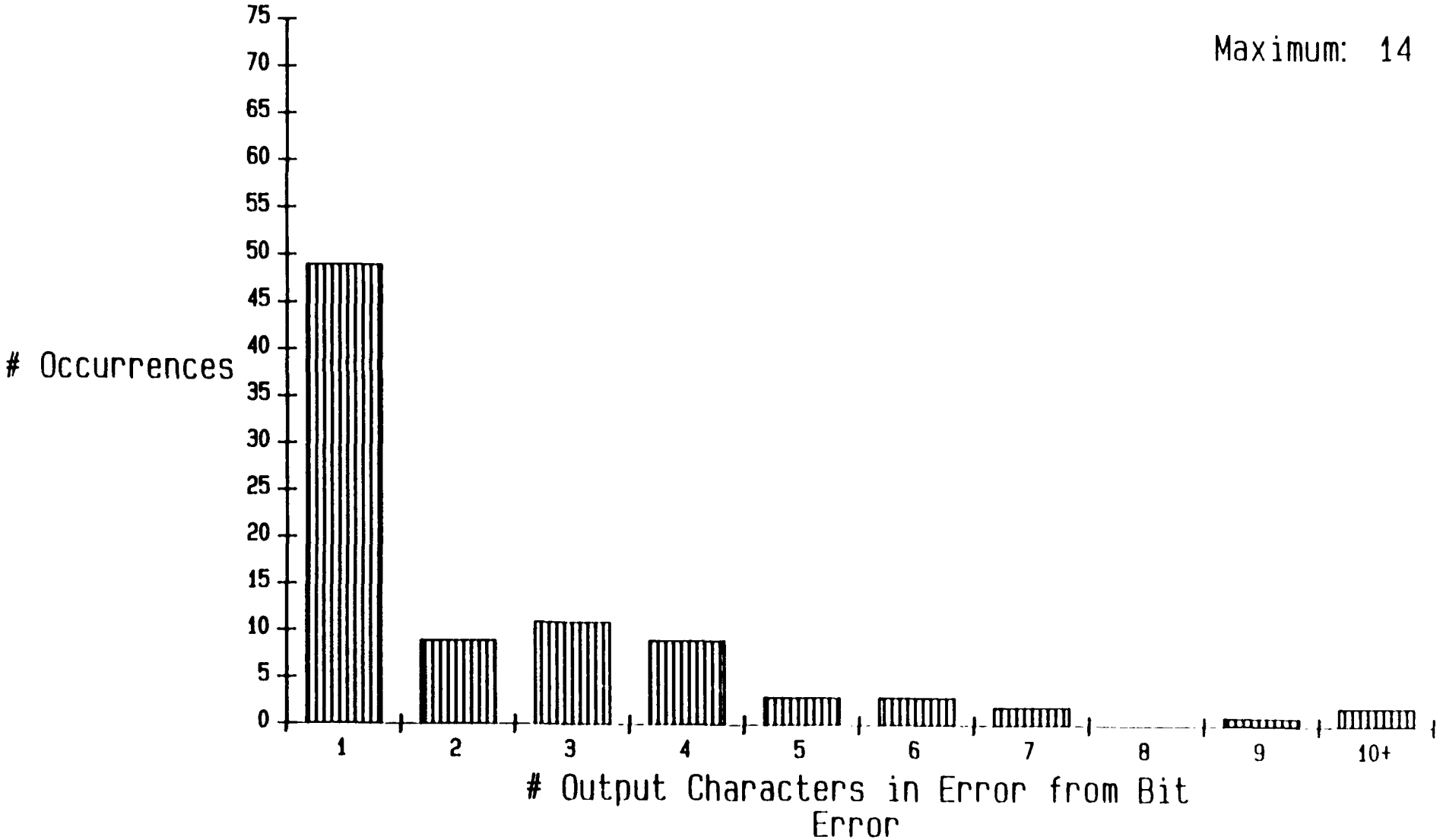


Figure 26. Distribution of Output Errors for Best 58 Character Huffman Code Using Narrative File IV as Training File

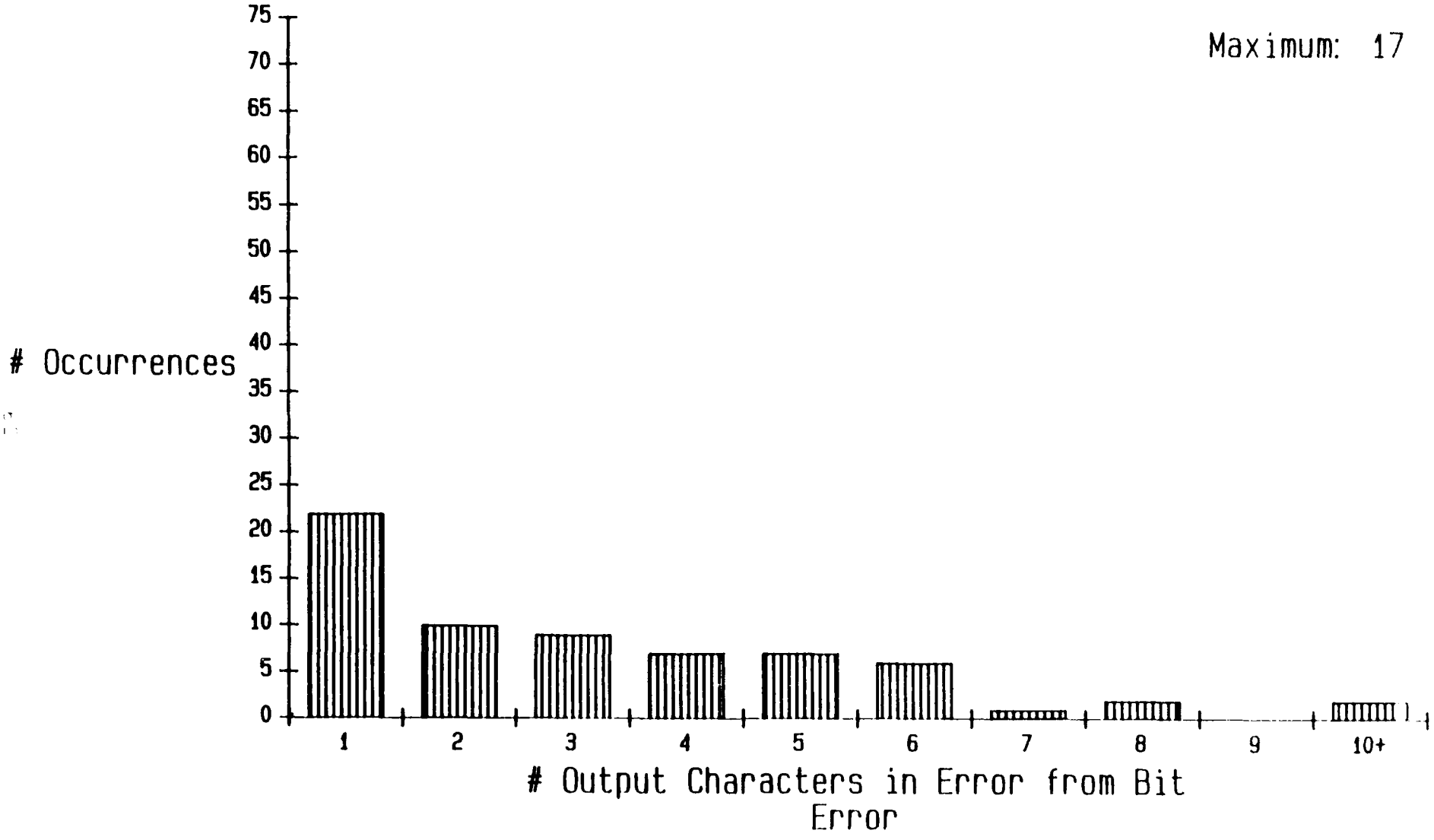


Figure 27. Distribution of Output Errors for Worst 58 Character Huffman Code Using Narrative File IV as Training File

sequences (35, 104, 65, and 90 for narratives I, II, III, and IV, respectively).

The best performing Huffman code found during the simulation had the distribution presented in figure 26 and occurred when narrative IV was used as a training file. This simulation only resulted in three output character sequences longer than 7 characters, one each of 9, 10, and 14 characters. The average length of an output sequence of character errors for this Huffman code was 2.4 output character errors.

It is worthwhile to compare the distribution shown for the best case in figure 26 with a distribution associated with the worst case shown in figure 27 which occurred for a code when narrative I was used as a training file. This distribution is nearly flat, extending beyond length 10 sequences. Indeed, there were 22 instances of output character error sequences of length 10 or more in the error simulation for this particular Huffman code.

Table 13 presents the Huffman code found through simulation using narrative file IV as a training file and providing the lowest average number of characters decoded in error per bit error (the code resulting in the distribution shown in figure 26).

Finally, we observe that for each narrative file as a training file and for each narrative file encoded and decoded in an error channel, there was a significant dependency of the average number of characters decoded in error per bit error on the particular Huffman code. There was usually at least a two-to-one difference in performance depending on the particular choices of "1"s and "0"s in the Huffman construction process. Recall that the number of bits-per-character depended only weakly on the narrative file using as a training file and not at all on the details of the Huffman code chosen. The results presented in this section show that both decode and output character error statistics are far more dependent on the construction process and narrative file statistics than compression results.

Comma-free Codes

In this section we first estimate the impact of errors on the performance of the different comma-free codes which can be constructed in two steps by choosing code words of length one. We then briefly discuss the impact of bit errors on the decoding errors for more general comma-free codes.

Comma-free Codes Constructed in Two Steps Using Length One Words

The error analysis discussed in this subsection is accomplished by obtaining the results for the two particular codes already discussed in the section describing comma-free codes. The codes were called the suffix-prefix code and the

TABLE 13. HUFFMAN CODE WORDS FOR NARRATIVE FILE IV WHICH PROVIDED THE BEST PERFORMANCE IN A CHANNEL WITH ERRORS

CHAR	CODE WORD	CHAR	CODE WORD
" "	01	@	1000101000010
!	10001010000000011	A	1110
"	111111110101	B	0001101
#	100010100000000100	C	11110
%	10001010000000000101	D	000011
'	1000101000000001	E	110
(	00100001	F	100011
)	00100000	G	111110
*	1111111101001	H	10000
+	0010101000	I	1010
,	1111110	J	111111110
-	10001011	K	001010101
.	0010001	L	000010
/	111111111	M	001001
0	00101011101	N	1001
1	001010110	O	1011
2	0010101111	P	001011
3	1111111011	Q	1000101001
4	001010111000	R	00000
5	100010100011	S	00010
6	00101011110010	T	0011
7	100010100000001	U	000111
8	100010100001	V	0001100
9	00101011110011	W	0010100
:	100010101100	X	111111100
;	111111101000	Y	1000100
<	10001010000000000	Z	1000101010
=	001010101	^	100010101101
>	10001010000001	~	10001010111

suffix-suffix code. After the error analysis has been carried out for these two codes, it is easy to argue that: (1) the analysis for the suffix-prefix code apply to all four codes using either "0" or "1" first and a suffix or prefix first and (2) the analysis for the suffix-suffix code applies to the four codes using "0" or "1" first and either both suffixes or both prefixes.

The first step in the analysis of each comma-free code is to calculate four probabilities:

P(X1) = the probability that a bit error leads to the deletion of a comma between two code words, i.e., to two code words being merged

P(X2) = the probability that a bit error leads to the movement of a comma relative to its true position if an error had not occurred

P(X3) = the probability that a bit error leads to the addition of a comma relative to those if an error had not occurred

P(X4) = the probability that a bit error leads to no change in the placement of the commas

The second step of the error analysis allows us to determine the impact of the bit errors upon character decoding. In particular, note the following:

(1) if a bit error leads to comma deletion then two characters are incorrectly decoded as a single character or not decodable

(2) if a bit error leads to comma movement then two characters and incorrectly decoded into two characters

(3) if a bit error leads to the insertion of a comma (always within the code word with the bit error) then one character will be incorrectly decoded into two characters

(4) if there is no change in the commas then one character will be incorrectly decoded into a single character

It follows that the average number of input characters in error or the average number of output characters in error can be estimated directly from the probabilities P(Xi), i = 1, 2, 3, 4.

Each bit error leads to an error in some character given by its character probability. For each character the impact of a bit error, assumed equally likely in each bit of the code word, can be calculated depending on the structure of the code word. To make this precise we introduce the following definitions:

p(c) = the probability of occurrence of character c

n(c) = the number of bits in the code word for c

n(c1) = the number of bits in the code word for c leading to the deletion of a comma

n(c2) = the number of bits in the code word for c leading to comma movement

n(c3) = the number of bits in the code word for c leading to the addition of a comma

n(c4) = the number of bits in the code word for c leading to no change in the comma

Then the probabilities P(Xi), i = 1, 2, 3, and 4, are calculated for the suffix-prefix code using the normal conditional probability procedure, namely

P(Xi) = sum over all characters of p(c)n(ci)/n(c) for i = 1, 2, 3, and 4

The required calculations for a particular assignment of the suffix-prefix code words to a 58-character set are easy but tedious.

Table 14 summarizes the calculations by character of the impact of bit errors for the probabilities of occurrence of characters in narrative file II. Similar results are expected for the remaining three narrative files. The first column contains the character probabilities of occurrence listed in descending order and the second column contains the comma-free code word associated to the character whose probability of occurrence is presented in the first column. Columns 3, 4, 5, and 6 present n(ci)/n(c), i = 1, 2, 3, and 4, respectively, for the character whose probability of occurrence is presented.

One observation should be made: the values of n(ci)/n(c) depend on the fine structure of the code words, and differ for words of the same length. The table has been constructed by assigning the code words with the most equal number of "0"s and "1"s to the highest probability character and the most unbalanced code words to the lowest probability characters. We illustrate this by discussing the summarized calculations for the words of length 8.

There are seven codes word of length 8 available. The two most unbalanced code words, namely, 01111111 and 00000001 have different n(ci)/n(c) values from the other six. This occurs because for these two codes an error in the first and last bit, respectively, turns the word into all "1"s and all "0"s, respectively, which lead to the deletion of the comma between the first and second words and the second and third words, respectively. Also, these two words only have one transition bit

TABLE 14. PROBABILITIES OF ERRONEOUS COMMA INSERTIONS OR DELETIONS DUE TO BIT ERRORS FOR THE SUFFIX-PREFIX COMMA-FREE CODE

PROBABILITY OF OCCURRENCE	CODEWORD	DELETE COMMA	COMMA MOVES	ADD COMMA	NO CHANGE
0.3171	01	1.00	0	0	0
0.0865	001	0.33	0.33	0	0.33
0.0677	011	0.33	0.33	0	0.33
0.0537	0011	0	0.50	0	0.50
0.0518	0111	0.25	0.25	0.25	0.25
0.0511	0001	0.25	0.25	0.25	0.25
0.0450	00111	0	0.40	0.20	0.40
0.0421	00011	0	0.40	0.20	0.40
0.0391	00001	0.20	0.20	0.40	0.20
0.0263	10000	0.20	0.20	0.40	0.20
0.0232	000111	0	0.33	0.33	0.33
0.0218	000011	0	0.33	0.33	0.33
0.0206	001111	0	0.33	0.33	0.33
0.0185	011111	0.17	0.17	0.50	0.17
0.0170	100000	0.17	0.17	0.50	0.17
0.0162	0000111	0	0.29	0.43	0.28
0.0141	0001111	0	0.29	0.43	0.28
0.0121	0011111	0	0.29	0.43	0.28
0.0101	0000011	0	0.29	0.43	0.28
0.0099	0111111	0.14	0.14	0.57	0.14
0.0082	1000000	0.14	0.14	0.57	0.14
0.0076	00001111	0	0.25	0.5	0.25
0.0058	00000111	0	0.25	0.5	0.25
0.0050	11100000	0	0.25	0.5	0.25
0.0039	00111111	0	0.25	0.5	0.25
0.0039	00000011	0	0.25	0.5	0.25
0.0034	01111111	0.13	0.13	0.63	0.13
0.0024	00000001	0.13	0.13	0.63	0.13
0.0023	000001111	0	0.22	0.56	0.22
0.0016	000011111	0	0.22	0.56	0.22
0.0014	000111111	0	0.22	0.56	0.22
0.0013	000000111	0	0.22	0.56	0.22
0.0013	000000011	0	0.22	0.56	0.22
0.0011	001111111	0	0.22	0.56	0.22
0.0010	011111111	0.11	0.11	0.67	0.11
0.0009	000000001	0.11	0.11	0.67	0.11
0.0008	0000011111	0	0.20	0.60	0.20
0.0007	0000111111	0	0.20	0.60	0.20
0.0005	0000001111	0	0.20	0.60	0.20
0.0005	0001111111	0	0.20	0.60	0.20

TABLE 14. PROBABILITIES OF ERRONEOUS COMMA INSERTIONS OR DELETIONS DUE TO BIT ERRORS FOR THE SUFFIX-PREFIX COMMA-FREE CODE (CONT.)

PROBABILITY OF OCCURRENCE	CODEWORD	DELETE COMMA	COMMA MOVES	ADD COMMA	NO CHANGE
0.0003	0000000111	0	0.20	0.60	0.20
0.0003	0011111111	0	0.20	0.60	0.20
0.0002	0000000011	0	0.20	0.60	0.20
0.0002	0111111111	0.10	0.10	0.70	0.10
0.0002	0000000001	0.10	0.10	0.70	0.10
0.0002	0000001111	0	0.18	0.64	0.18
0.0002	0000011111	0	0.18	0.64	0.18
0.0002	0000111111	0	0.18	0.64	0.18
0.0002	0000000111	0	0.18	0.64	0.18
0.0002	0001111111	0	0.18	0.64	0.18
0.0001	0000000011	0	0.18	0.64	0.18
0.0001	0011111111	0	0.18	0.64	0.18
0.0001	0000000011	0	0.18	0.64	0.18
0.0001	0111111111	0.09	0.09	0.73	0.09
0.0000	0000000001	0.09	0.09	0.73	0.08
0.0000	0000001111	0	0.17	0.66	0.17
0.0000	0000011111	0	0.17	0.66	0.17
0.0000	0000000111	0	0.17	0.66	0.17

position which is not an edge position, and only the second to an edge bit can be in error without changing any commas; while for all other words, either of the transition bits can be in error without changing any commas. Finally, these two code words have five interior positions within a string of "1"s or "0"s while all the other words only have four; errors in these bit positions lead to an additional comma within the word with the bit error.

For the assignment of suffix-prefix codes to characters described above and summarized by table 11, the following statistics were obtained:

$$P(X1) = .42$$

$$P(X2) = .21$$

$$P(X3) = .16$$

$$P(X4) = .21$$

Note that about three-quarters of the contribution to P(X1) is that provided by the code word "01" assigned to the blank character with probability of occurrence 0.317124.

The average number of coded characters decoded in error per bit error is given by

$$(.42)(2) + (.21)(2) + (.16)(1) + (.21)(1) = 1.63$$

The average number of output characters which are incorrect per bit error is given by

$$(.42)(1) + (.21)(2) + (.16)(2) + (.21)(1) = 1.37.$$

These values are obtained by treating words too long to be decoded because they exceed the longest word assigned one of the 58 characters as being incorrectly decoded. (Such characters could be decoded into a 59-th character indicating an error has occurred.) To distinguish these cases from the cases when the erroneous words arising through misplacement of commas can be decoded into one of the 58 characters to which code words have been assigned would require more delicate arguments depending on the lengths and structure of the code words for the characters preceding and following the one in error.

The calculations presented clearly indicate that the performance of the prefix-suffix code in an error channel is considerably better than the performance of any Huffman code that we found, so that these more delicate calculations are not necessary. And, this improved performance was obtained by paying an insignificant penalty in compression (or equivalently, thruput).

A two step comma-free code construction using a one-bit prefix and a one-bit suffix, no matter what choices are made

leads to code words either of the form

....1... or ...10...

- th each code word containing at least one "1" and the "1" [illegible] these codes can be obtained from the other by interchanging "1"s and "0"s. If this were done to the assignment of code words the same probabilities p(c)/n(c) would be obtained as for the code word assignment before the interchange. Thus all the prefix-suffix codes using a one-bit prefix and a one bit suffix would for these assignments have the same error statistics.

We turn now to estimating the impact of errors in the suffix-suffix code discussed earlier. Recall that the code words for this code have the structure 01...10...0 with at least one "1". This code differs from the suffix-prefix code in that the impact of an error in the first position of a code word depends on the ending of the previous code word.

We treat the impact of the first bit and the second bit in the next two paragraphs and then complete the analysis in tabular form. However, the assignment of code words to characters as specified in table 15 is the basis for all of the calculations. We only carry out the calculations for the probabilities of occurrence of the characters in narrative file II. Similar results would be obtained for the probabilities of occurrence of the characters in the other three narrative files.

Table 15 summarizes the impact of errors from the third bit through the end of a word. These errors either lead to the addition of a comma, or no change in the commas, depending on whether or not the bit error changes a "1" to a "0" within a string of "1"s and whether or not an error changes a "0" to a "1" within a string of "0"s. We now turn to evaluating the impact of errors in the first two bits

An error in the first bit leads to the movement of a comma or the deletion of a comma depending on whether the first code word ends in "0" (0.51 from table 15) or ends in "1" (0.49). Therefore,

(1) the probability that a bit error in the first bit leads to the movement of a comma is given by:

(probability that a code word ends in "0")(sum over characters of p(c)/n(c)) = (0.51)(.31) = .158

(2) the probability tnat a bit error in the first bit leads to the deletion of a comma is given by:

(probability that a code word ends in "1")(sum over characters of p(c)/n(c)) = (0.49)(.31) = .152

An error in the second bit leads to the deletion of a comma or the addition of a comma, depending on whether or not the

TABLE 15. PROBABILITIES OF ERRONEOUS COMMA INSERTIONS DUE TO BIT ERRORS FOR THE SUFFIX-SUFFIX COMMA-FREE CODE

PROBABILITY OF OCCURRENCE	CODEWORD	ADD COMMA	NO CHANGE
0.3171	01	0	0
0.0865	010	0	0.33
0.0677	011	0	0.33
0.0537	0100	0.25	0.25
0.0518	0110	0.25	0.50
0.0511	0111	0.25	0.25
0.0450	01000	0.40	0.20
0.0427	01100	0.20	0.40
0.0391	01110	0.20	0.40
0.0263	01111	0.40	0.20
0.0232	010000	0.50	0.17
0.0218	011000	0.33	0.33
0.0206	011100	0.33	0.33
0.0185	011110	0.33	0.33
0.0170	011111	0.50	0.17
0.0162	0100000	0.57	0.14
0.0141	0110000	0.43	0.28
0.0121	0111000	0.43	0.28
0.0101	0111100	0.43	0.28
0.0099	0111110	0.43	0.28
0.0082	0111111	0.57	0.14
0.0076	01000000	0.62	0.13
0.0058	01100000	0.5	0.25
0.0050	01110000	0.5	0.25
0.0039	01111000	0.5	0.25
0.0039	01111100	0.5	0.25
0.0034	01111110	0.5	0.25
0.0024	01111111	0.63	0.13
0.0023	010000000	0.67	0.11
0.0016	011000000	0.55	0.22
0.0014	011100000	0.55	0.22
0.0013	011110000	0.55	0.22
0.0013	011111000	0.55	0.22
0.0011	011111100	0.55	0.22
0.0011	011111110	0.55	0.22
0.0009	011111111	0.67	0.11
0.0008	0100000000	0.70	0.10
0.0007	0110000000	0.60	0.20
0.0005	0111000000	0.60	0.20
0.0005	0111100000	0.60	0.20
0.0003	0111110000	0.60	0.20
0.0003	0111111000	0.60	0.20
0.0002	0111111100	0.60	0.20

TABLE 15. PROBABILITIES OF ERRONEOUS COMMA INSERTIONS DUE TO BIT ERRORS FOR THE SUFFIX-SUFFIX COMMA-FREE CODE (CONT.)

PROBABILITY OF OCCURRENCE	CODEWORD	ADD COMMA	NO CHANGE
0.0002	0111111110	0.60	0.20
0.0002	0111111111	0.70	0.10
0.0002	01000000000	0.73	0.09
0.0002	01100000000	0.64	0.18
0.0002	01110000000	0.64	0.18
0.0002	01111000000	0.64	0.18
0.0002	01111100000	0.64	0.18
0.0001	01111110000	0.64	0.18
0.0001	01111111000	0.64	0.18
0.0001	01111111100	0.64	0.18
0.0001	01111111110	0.64	0.18
0.0000	01111111111	0.73	0.09
0.0000	010000000000	0.75	0.08
0.0000	011000000000	0.67	0.17
0.0000	011100000000	0.67	0.17

second bit in error was the only "1" in the code word. Therefore,

(1) the probability that a bit error in the second bit leads to the deletion of a comma is given by:

$$p(2)/2 + p(3)/3 + \ldots + p(12)/12 =$$

$$\begin{aligned} &(0.3171)(1/2) \\ &+ (0.0865)(1/3) \\ &+ (0.0537)(1/4) \\ &+ (0.0450)(1/5) \\ &+ (0.0232)(1/6) \\ &+ (0.0162)(1/7) \\ &+ (0.0076)(1/8) \\ &+ (0.0023)(1/9) \\ &+ (0.0008)(1/10) \\ &+ (0.0002)(1/11) \\ &+ (0.0000)(1/12) \end{aligned}$$

$$= 0.197 \text{ [rounded to three places]}$$

where p(2), p(3), ..., p(12) are the probabilities of occurrence of the character which has been assigned a word of length two with a single "1", a word of length three with a single "1", ..., a word of length twelve with a single "1", respectively (see table 15).

(2) the probability that a bit in the second bit leads to the addition of a comma is given by:

$$\text{(sum over characters of } p(c)/n(c)) - [p(2)/2 + p(3)/3 + \ldots + p(12)/12] =$$

$$.312 - .197 = .115$$

where p(2), p(3), ..., p(12) are as above.

The probability that a bit error leads to the addition of a comma through changing other than the first or second bit is .18 from summing the data presented in column 3 of table 15. The probability that a bit error leads to no change in the commas through changing other than the first or second bit is .20 from summing the data presented in column 4 of table 15.

For the assignment of suffix-suffix codes to characters described above and summarized by table 15, the following summary statistics follow from combining the estimates which have been obtained:

$$P(X1) = .35$$

$$P(X2) = .16$$

$P(X3) = .29$

$P(X4) = .20$

Note that about three-quarters of the contribution to P(X1) is that provided by the code word "01" assigned to the blank character with probability of occurrence 0.317124.

The average number of coded characters decoded in error per bit error is given by

$$(.35)(2) + (.16)(2) + (.29)(1) + (.20)(1) = 1.51$$

The average number of output characters which are incorrect per bit error is given by

$$(.35)(1) + (.16)(2) + (.29)(2) + (.20)(1) = 1.45.$$

The statistics developed for the particular suffix-suffix code applies to the other suffix-suffix code and to both prefix-prefix codes. Observe that if "0" is chosen as a suffix first and then "1", the code words have the structure

10...01...1 with at least one "0" and always starting with "1"

This code is obtained from the one analyzed by interchanging the roles of "0" and "1" so it will have the same statistics as the code analyzed, provided that the code words assigned to the characters are those obtained by interchanging "0"s and "1"s in the assignment made previously. Observe that if "1" is chosen as a prefix, and "0" as a prefix, the code words have the structure:

0...01...10 with at least one "1" and always ending with "0"

These code words are simply the mirror images of the words in the suffix-suffix code analyzed. Assign these code words to characters by taking mirror code words to those assigned for the suffix-suffix code. Then, the first position analysis, which depended on the ending of the prior word, applies to the last bit of the prefix-prefix code and the beginning of the next code word ending in "0" or "1". The second bit analysis applies to the second to last bit. The analyses conducted before clearly apply to the remaining bits. It follows the statistics will be the same for this prefix-prefix code.

The remaining prefix-prefix code is obtained from the one just discussed by interchanging "1"s and "0"s; therefore, it will also have the same statistics, provided (once again) that the assignment of code words to characters is obtained by interchanging "1"s and "0"s in the above assignment.

Comma-free Codes Constructed Using Other Than Length One Words

The results presented in the last section were for the two simplest kinds of comma-free codes. In this section, it is shown that more complicated phenomena can occur leading to more than two character decoding errors as a result of a single bit error. The results indicate that as the number of steps in the comma-free code construction process increases without limit the number of character decoding errors probably increase without limit. However, we have not succeeded in exhibiting this for a sequence of comma-free codes involving the use of longer and longer construction processes.

Some additional terminology is needed to facilitate the discussion of general comma-free codes. Let

k denote the kernel of the code under construction

p(i) , i = 1, 2, ... denote the prefixes used in the code under construction

s(j), j = 1, 2, ... denote the suffixes used in the code under construction

The most general comma-free codes have not been discussed in this manuscript and have not been analyzed in this study. We impose the following additional conditions on the codes under discussion:

(1) k = "0" or "1"

(2) both "0" and "1" are used as either prefixes or suffixes

(3) the length of the prefix or suffix used in k-th construction step is less than or equal to the length of the prefix or suffix used in the (k+1)-th construction step. (These restrictions may not be necessary for carrying out an error analysis similar to that presented, but they are convenient and probably do not exclude any codes that are of interest for data compression.)

The results presented in the last section can be generalized to an important family of comma-free codes, which we call exhaustive codes. A comma-free code is called exhaustive if for each of the steps in the code construction process the code word chosen as either a prefix or suffix is the shortest code word possible. For example, referring to tables 7,8,9, and 10, the codes (1,1), (1,1,2), (1,1,2,3), and (1,1,2,3,3) are exhaustive and the codes (,1,3), (1,1,3,3), (1,1,2,4), and (1,1,4) are non-exhaustive codes.

For an exhaustive comma-free code, a single bit error can lead to at most two characters decoded in error. For non-exhaustive comma-free codes, it may happen that a single bit error leads to more than two characters decoded in error. To

establish this result, consider (1) an incoming sequence of bits as a sequence of kernels, prefixes, and suffixes, and (2) the comma-insertion algorithm (after the first step) as deleting commas between the kernels, prefixes, and suffixes. Now, let us discuss the potential impact of a single bit error occurring in a kernel or in a prefix or suffix of the code words. In particular, we wish to discuss how a error can impact the comma deletion process between two words which are error free.

Let us denote the word with a bit error by use of "^". Consider, the incoming sequence of binary bits parsed into codewords as follows:

w(1)w(2)w^(3)w(4)w(5).

Under what conditions will the comma separating w(1) and w(2) or the comma between w(4) and w(5) be altered as a result of a bit error somewhere in the codeword w(3)? Each of these words is constructed from the kernel and prefixes and suffixes, as described above.

For the comma between w(1) and w(2) to be erased by the comma-insertion algorithm, the prefix or kernel beginning w(2) must be transformed into a suffix through a bit error in w(3). Since none of bits in w(2) are in error, this can only happen if the addition of bits to the bits of w(2) has created a suffix used in the construction process; i.e., there exists a code word of shorter length in the code than some suffix in the code. This means the comma-free code is non-exhaustive.

For the comma between w(4) and w(5) to be erased by the comma-insertion algorithm, the suffix or kernel ending w(4) must be transformed into a prefix through a bit error in w(3). Since none of the bits in w(4) are in error, this can only happen if the addition of bits to the bits of w(2) has created a prefix used in the construction process; i.e., there exists a code word of shorter length in the code than some prefix in the code. This means the comma-free code is non-exhaustive.

It is clear that one could improve upon the results by examining the non-exhaustive codes to see if either of the above phenomena can occur for a particular selection of prefixes or suffixes. This is relatively straightforward for any family of codes constructed using mostly short length prefixes and suffixes and no more than six construction steps. This is because the analysis is carried out by examining only those code words less than the longest suffix or prefix used in the construction. We illustrate this by considering the suffix-prefix codes of the form (1,1,3).

For kernel "0", suffix "1", prefix "0", there would be two three letter code words available, namely "001" and "011", to be selected as either a suffix or prefix. The only length two code words are "00" and "01". Note if "011" is chosen as a prefix, there is no one bit which can be combined with "00" or "01" from

the left to create it; if instead "001" is chosen as a prefix "0" could be combined with "01" to obtain it. However, under the same conditions if "011" is chosen as a suffix, then a one-bit "addition" to "01" on the right leads to the chosen suffix; if instead "001" is chosen as a suffix then again a one-bit "addition" to "00" on the right leads to the chosen suffix. Thus, for three of the four constructions considered, one bit error could lead to three characters decoded in error. Note, that for these codes, there has to be a very special combination of words and very particular bit errors to lead to more than two character decoding errors as a result of a single bit error.

If we consider when a single bit error could lead to four character decoding errors, similar reasoning to that presented above would lead to the necessity that two code words of length 2 plus one or more bits would need to be a suffix or prefix; i.e., the code would need to involve a suffix or prefix of length 5 or more. Hence, for the four non-exhaustive codes discussed in this section, namely (1,1,3), (1,1,3,3), (1,1,2,4), and (1,1,4), a single bit error can never lead to four or more character decoding errors.

SUMMARY

The present code consists of a parity bit, 5 information bits, and a stop bit. Our discussion of generalized Baudot codes suggested that a code using 4 information bits could replace the 5 information bit code now being used. We suggest a parity bit, 4 information bits, and 1/2 bit for stops (i.e., a stop bit for every 8 information bits). The number of bits for the new code is given by:

$$4.5 + (4.5/4)(1.5) = 6.2 \text{ bits-per-character}$$

The data compression provided by this code would be

$$7/6.2 = 1.13$$

The more complicated encoding and decoding associated with comma-free codes does promise some additional compression. However, a mechanism to allow receiver synchronization in the absence of stop bits needs to be identified, and the incorporation of error correction codes requires care. Note, one cannot just add parity bits operating on code words, because in their presence the comma insertion algorithm would break down. Error correction information must be carried by comma-free code words. We suggest that it be incorporated into the encoding of the end-of-line character. One would map the error correction information into a set of long code words which could provide correction for the line and end-of-line indication by its presence.

It is always possible to superimpose an error correction code on the serial bit sequence before transmission and then utilize it for error correction before beginning the comma-free decoding

process which begins with the comma-insertion algorithm.

If error detection and correction bits were kept to the same ratio of information bits to non-information bits as the code discussed above, we would need

4 + 1.5 = 5.5 bits-per-character

This would translate into a data compression ratio of

7/5.5 = 1.37

This compression ratio is probably the best that could be accomplished using comma-free codes and character encoding based on probabilities of occurrence of the characters.

More powerful encoding techniques would use conditional probability of occurrences of characters. The suggested approach would be to create a table of comma-free code words for characters depending on the previously transmitted one or two characters. The encoding process would be reinitialized with the beginning of each word. As a further aid to the identification of the beginnings of words, it might prove desirable to always encode spaces in the same way by reserving some particular code word for spaces. It is recommended that the use of conditional probabilities of character occurrence rather than probabilities of occurrence and the appropriate error detection and correction coding for use with the compression code be analyzed in a follow-on effort. Such an approach promises considerable additional compression over that shown for any of the cases investigated in this report.

LIST OF REFERENCES

1. Huffman, D., "A Method for the Construction of Minimum Redundancy Codes", Proceedings of the Institute of Radio Engineers, Vol. 40, pp. 1098-1101, September 1952.

2. Scholtz, R., "Codes with Synchronization Capability", IEEE Transactions on Information Theory, Vol. IT-12, No. 2, April 1966.

3. Scholtz, R., "Maximal and Variable Word-Length Comma-Free Codes", IEEE Transactions on Information Theory, Vol. IT-15, No. 2, March 1969.

END
DATE
FILMED

2-9-88